AFIT/GCS/ENG/91D-25

DTIC
ELECTE
DEC 27 1991
S C D

AD-A243 633

A CACHE DESIGN TO EXPLOIT STRUCTURAL LOCALITY

THESIS

Curtis M. Winstead, Captain, USAF

AFIT/GCS/ENG/91D-25

Approved for public release; distribution unlimited

91-19021

91 12 24 041

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE
A CACHE DESIGN TO EXPLOIT STRUCTURAL LOCALITY

5. FUNDING NUMBERS

6. AUTHOR(S)
Curtis M. Winstead

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/91D-25

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited (Central Processing Unit)

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A design and VHDL implementation of a content-addressable memory (CAM) to exploit structural locality is the subject of this research. The concept of structural locality is that memory locations are referenced in the same order as they were previously referenced. Therefore, if memory locations that exhibit structural locality can be made available to the CPU through a fast data store, an increase in speed of the computer system can be realized. The CAM's purpose is to store memory references in the order they were used by the CPU and prefetch these locations to a smaller on-chip cache. The CAM uses a FIFO circular buffer algorithm to store the memory references. When the CPU references a location that is stored in the CAM, the CAM prefetches memory locations in a FIFO manner, thus allowing the on-chip cache to capture structural locality into its memory. Basic digital logic circuits were implemented in VHDL and were the building blocks for the cache model. From these, the controller, which controls the prefetching of structural locality, was then integrated onto the chip model containing a fully-associative CAM array.

14. SUBJECT TERMS	15. NUMBER OF PAGES
VHDL, Content-Addressable Memory, Cache, Structural Locality	230
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

AFIT/GCS/ENG/91D-25

A CACHE DESIGN TO EXPLOIT STRUCTURAL LOCALITY

THESIS

Presented to the Faculty of the School of Engineering

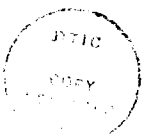
of the Air Force Institute of Technology

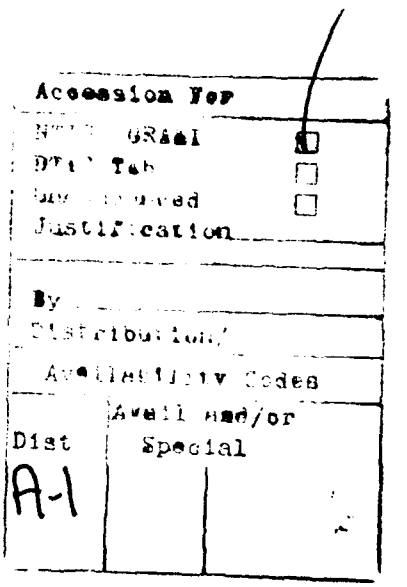
Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Science





Curtis M. Winstead, B.S.

Captain, USAF

December 1991

Approved for public release; distribution unlimited

Acknowledgments

This thesis document represents the hardest, most consistent work I have labored through during my AFIT tour. Designing a content-addressable memory chip for use in a modern computer was both exciting and challenging. Both of these factors made this a very rewarding experience.

Many deserve thanks for helping me through this trying time. I thank God foremost for listening to my prayers and encouraging me through his Word. These words were especially helpful to me: "The plans of the diligent lead surely to advantage, But everyone who is hasty comes surely to poverty" (Proverbs 21:5). When the workload seemed too great, He reminded me that diligence pays off. He provided me the strength to strive to completion.

To my thesis advisor, Major Bill Hobart, I thank for the undying support he gave me. His door was always open for me and he was willing to help whenever I needed it. I thank him for the freedom to approach the project in my own way and for the guidance that kept me on track. I sincerely thank my committee members for their help. Captain Mark Mehalic greatly assisted me with VHDL and the ZYCAD environment. He unloaded a burden from my shoulders at the inception of this project with the time he spent with me discussing VHDL. He made this seemingly monumental task look like a manageable project. Dr. Frank Brown helped me tremendously in the writing of this document. I asked him to be on my committee because of his background in computer hardware, but he quickly became an asset from the writing standpoint. I just regret not giving him drafts of this document sooner so he could have helped me even more.

Finally, I thank my family for their patience and understanding throughout this year and a half. I am so thankful for my wife, Theresa, and the support and love she offered me. She never lost sight of my goals; she worked hard for me so I could work hard

for our future. I thank my daughters, Laura and Christi, that though they were very young (5 and 4 years old, respectively, when I graduated), they understood that daddy had to work very hard. To my wonderful, loving family I dedicate this thesis. I love you.

Curtis M. Winstead

Table of Contents

List of Figures

List of Tables

Table Page

Abstract

This research involved the design and VHDL implementation of a content-addressable memory to exploit structural locality. The concept of structural locality is that memory locations are referenced in the same order as they were previously referenced. Therefore, if memory locations that exhibit structural locality can be made available to the CPU through a fast data store, an increase in speed of the computer system can be realized.

A content-addressable memory (CAM) cache was used to supply data to an on-chip cache that acts as this fast data store. The CAM is described in this study and is a member of a two-cache memory hierarchy. Its purpose is to store memory references in the order they were used by the CPU and prefetch these locations to a smaller on-chip cache for fast processing. The CAM emulates an LRU stack by using a FIFO circular buffer algorithm to store the memory references. When the CPU references a location that is stored in the CAM, the CAM prefetches memory locations in a FIFO manner, thus allowing the on-chip cache to capture structural locality into its memory.

A fully-associative content-addressable memory was used in this study. This type of memory allows its contents to be searched in parallel. When a search is successful, the contents of the memory location are read and a top-of-stack pointer is incremented to read successive memory locations from the CAM array. A bottom-up design approach was used to build this cache. Basic digital logic circuits were implemented in VHDL and were the building blocks for the model. Using these basic components, the major components that make up the controller were made. The controller, which controls the prefetching of structural locality, was then integrated onto the chip model containing a fully-associative CAM array.

A CACHE DESIGN TO EXPLOIT STRUCTURAL LOCALITY

I. Introduction

Overview

A computer's main memory is very important in the support of the operating systems and the users. The technologies of the 1950s and 1960s made it very expensive to have an adequate amount of this main memory. Conversely, secondary storage is relatively inexpensive and has a much greater capacity than main memory. Unfortunately, secondary storage has a much slower access speed. As a result, the idea of a memory hierarchy was introduced. Initially, the hierarchy consisted of main memory and secondary storage. Main memory was used to store the instructions and data of an executing program, while secondary storage held programs and data that were not immediately needed. Since main memory was more expensive, it was generally smaller than secondary storage. (4:188)

Cache memory was introduced in the 1960s and formed additional levels of the memory hierarchy. Cache memory is very fast storage designed to increase the speed of running programs. The ideal memory system would be one that holds infinitely large files, has an infinitesimal access time, and is free! Unfortunately, this is not possible, but cache memory is the next best thing. Using the concept of locality (explained below), caches simulate a larger memory by storing data that are frequently used by the central processing unit (CPU). Although a zero effective access time is not technologically possible, caches are much faster than main memory and secondary memory. Also, since

caches are composed of faster, and thus more expensive memory, only relatively small caches are economical.

A side effect of memory hierarchies, to which caches add another level, is data shuttling. Shuttling occurs when data are transferred from one level of hierarchy to another. This decreases the efficiency of the CPU (4:188). If shuttling can be decreased and access time to memory can be significantly reduced, the CPU could be much more productive. Cache memories are used exclusively to reduce access time. This thesis will focus on caching between the CPU and main memory.

Although caches are small, the memory hit-ratio on these caches can be extremely high due to the concept of locality. The most widely recognized aspects of locality are *spatial* and *temporal*. Spatial locality implies that if a memory location is referenced, then it is likely that the memory locations nearby in the virtual memory address space will also be referenced. Temporal locality means that if a memory location is referenced, then it is likely to be referenced again in the near future. Exploiting locality results in an increase in efficiency of the CPU and faster turnaround of executing programs.

In addition to the well known spatial and temporal aspects of locality, Hobart has identified a third aspect, which he has called structural locality (9). This type of locality is defined as the tendency of an executing program to reference memory locations in the same order in which they were previously referenced. Thus, if memory references were placed on a stack in the order in which they were referenced, a reference to a particular memory location in the stack increases the probability of subsequent accesses to memory locations immediately above it in this stack.

A software model using VHDL (Very High Speed Integrated Circuit (VHSIC) Hardware Description Language) of a content-addressable memory (CAM) with an integrated structural locality cache (SLC) controller will allow further investigation into

structural locality with much greater flexibility than with a hardware prototype. The VHDL model can be changed to test a desired behavior, whereas an actual realization in hardware is much more difficult to alter. Although CAMs have previously been designed, this researcher has not found one that has been modeled using VHDL.

Problem Statement

The problem addressed in this thesis is the design of a prefetching memory system to exploit the aspects and benefits of structural locality. The goal of this thesis effort was to design a main system cache that supports structural locality prefetching to a smaller on-chip cache.

Scope/Limitations

This thesis includes developing a behavioral and structural description of an SLC controller integrated into a CAM cache. Therefore, the structure of a CAM cache with an integrated SLC controller was modeled using VHDL. The gate level was the lowest level modeled.

The implementation and fabrication of the cache were beyond the scope of this thesis. However, this thesis should enable fabrication of an actual CAM chip with an integrated SLC controller by a follow-on thesis.

Approach

The product of this thesis was a design of an SLC controller integrated with a fully-associative cache. The controller controls the prefetching of the cache contents immediately above the currently referenced CAM location and the writing of data into the CAM array.

A bottom-up approach for modeling the CAM cache was used. With the CAM as described by DeCegama (3:82-88), a gate-level structural description of a CAM cell was

modeled. The cell was tested to determine if its features were acceptable. After verifying that the cell performed as expected, copies of the cell were integrated to form the CAM array. The CAM array was then tested to verify its expected behavior.

After the VHDL model of the CAM array was complete, design of the SLC controller began. Again, a bottom-up approach was used. First, the functions of the controller were defined. Next, the structure of the controller was determined in the form of a schematic diagram. Finally, the structures of this controller were integrated with the CAM array.

To test and verify the CAM and controller as described by VHDL, the external hardware, with which the cache will interface, was modeled. The external hardware includes the CPU main memory. The CPU was modeled using actual virtual memory address traces. For a virtual memory reference resulting in a main cache miss, an appropriate delay simulated the fetching of the cache line from main memory. After testing all pieces of hardware individually, they were connected to form a closed system. This system was then used to test and verify the expected behavior of the CAM chip model.

Thesis Contents

Chapter 2 contains an overview of computer memory systems. Associative memories, in particular, are discussed. The various types of content addressable memories are presented. They include bit-slice, byte-slice, word-slice, and fully-associative memories. Several implementations of these types are shown and the advantages and disadvantages of CAMs are discussed. Most of the CAM cells in the literature are described at the transistor level; therefore, many transistor level designs are presented.

Chapter 3 describes the VHDL implementation of the main CAM cache (MCC) for the proposed memory subsystem. The design of the cache is described in detail. The methodology used to design the MCC is discussed as well as the building of the MCC

model in a hierarchical manner in VHDL. Since the cache was designed from the bottom up, the functionality of each component making up the cache is explained.

Chapter 4 discusses the testing and performance characteristics of the main CAM cache. It describes the overall behavior of the MCC as well as an analysis of the timing constraints during each activity the MCC performs. Suggestions on possible improvements to the MCC are proposed and areas where potential space savings can be made during fabrication are discussed. Other issues to consider during the hardware implementation of the chip are reviewed.

Finally, Chapter 5 summarizes this thesis effort with conclusions and recommendations for further study.

II. Background

Introduction

A major goal for computer architects is to increase the speed of executing programs. "One such technique is the use of memory hierarchies - in particular the cache store concept. This approach to computer memory speedup has been well proven in the large processor situations" and "is also applicable to smaller machines where the economic constraints are more severe" (1:75). As computer technology has grown in complexity, so have computer applications. As these applications become more complex, the need for speed becomes increasingly important. To increase speed and "in order to minimize bus traffic, cache memory is often placed between the processor and the shared bus" (17:218).

Since caches are expensive and small, many studies have tried to determine an optimal size for a cache to get the highest hit ratio possible. "These studies suggest that the single most important factor for improving the cache hit ratio is the size of the cache memory" (17:219). Quinones suggests that a variable-size cache can allow many cost/performance goals to be reached (17:219). Ackland (1:76) points out that the optimal cache size depends on the processor architecture and the software environment that is being used. Ackland also found through simulation results that "effective speed up can be gained from buffers ranging in size from 256 words to 1024 words" (1:76).

Content-addressable memories are used as caches and have been investigated since 1956 (15:453). A CAM's purpose is to locate data by its contents rather than by its address, thereby increasing the speed at which memory is accessed. Memory access to data is accomplished differently in CAMs than with conventional random access memories (RAMs). Data access in RAMs is done by decoding the address and then fetching the

data. In contrast, data in CAMs are located by their content. Minker (15:453) states that "the retrieval of any one item in such a store would be accomplished by performing a content search on all registers in parallel with but a single operation." Not only is a search performed based on contents, but magnitude relationships such as less than, between limits, next higher/lower, similarity, proximity, not equal, or minimum/maximum value can also be accomplished (2:52). Addressing by content eliminates the need for such operations as scanning, sequential searching, and counting.

CAMs help to overcome what is known as the "von Neumann bottleneck." This bottleneck is caused by the communications between the CPU and the memory. "To reduce the traffic on this data path, and thereby increase system performance, one may add limited processing capabilities to the memory side of the bottleneck" (8:537). Content-addressable memories contain these limited processing capabilities, thereby reducing the von Neumann bottleneck effect. "Associative processors go a step further, eliminating the bottleneck entirely by performing both data storage and processing functions in a single unit." (8:537)

Target Architecture

Hobart (9) proposed the SLC memory subsystem shown in Figure 1. Two CAMs are used in this three-level memory hierarchy. The first CAM, the main CAM, interfaces between main memory and the on-chip cache. The second CAM is the on-chip cache that interfaces between the main CAM and the CPU. This author has not found this type of architecture implemented with CAMs.

The purpose of the two-CAM system is to take advantage of structural locality (explained in Chapter 1). The main CAM emulates the top portion of a least recently used (LRU) stack thereby mapping the temporal locality of the virtual memory references

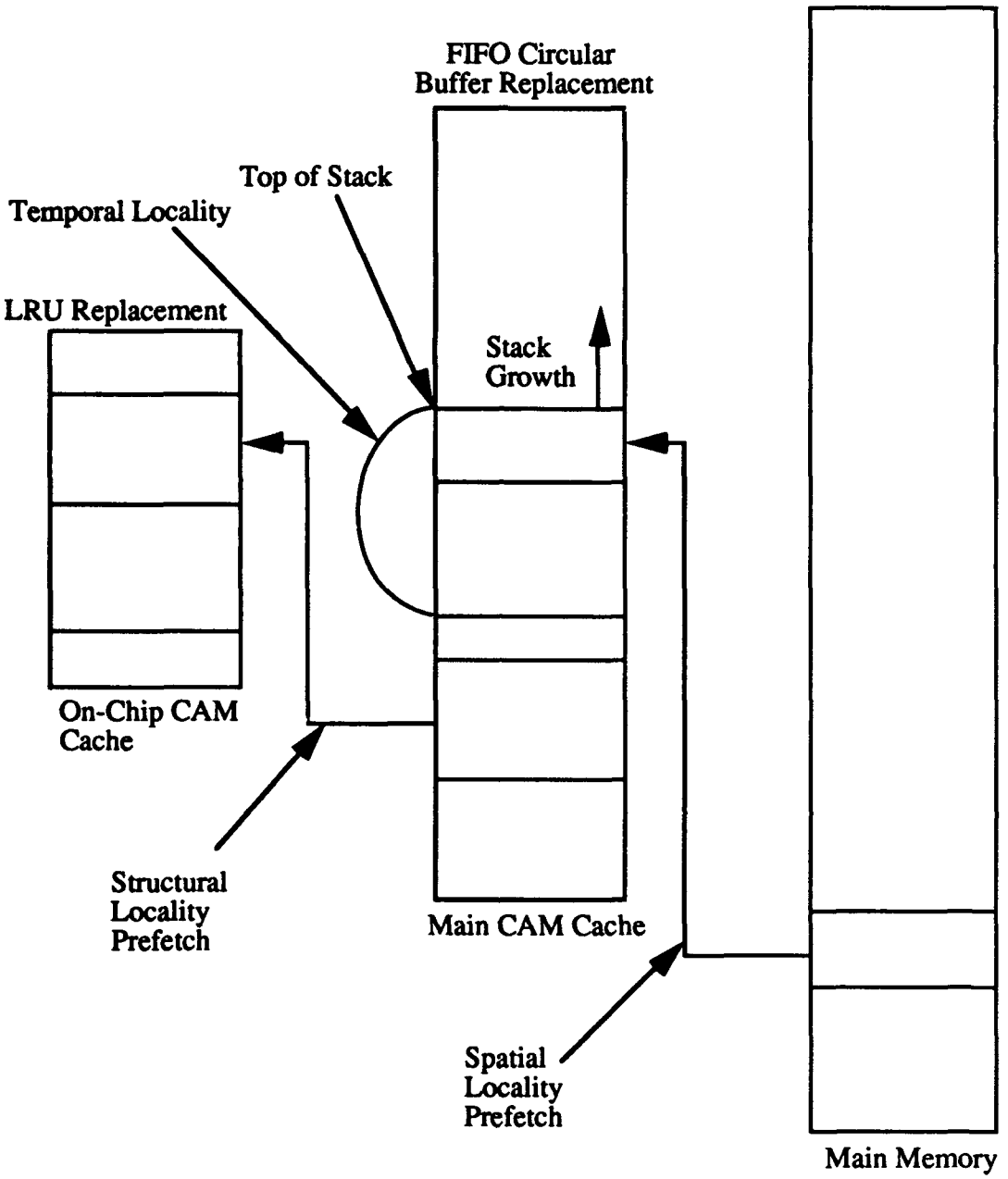


Figure 1. Proposed Memory Subsystem Design (9:98)

into spatial locality within the main CAM. Structural locality is then exploited by spatially prefetching from the main CAM into the on-chip cache. (9:97)

Hobart chose the main CAM to be fully associative because "if the simulated LRU stack can be realized in a cache with full associativity, then any position in the stack

can be referenced in constant time, and the references immediately above the stack position accessed can be prefetched into a smaller faster on-chip cache..." (9:96). The on-chip cache, or the SLC, requires that the main cache be fully associative to locate any memory location on the stack. This enables prefetching to the SLC based on structural locality. The main CAM cache is written to with a circular buffer replacement algorithm. Therefore, when this cache prefetches to the SLC, the data references received by the SLC will be in the order in which the main CAM cache received them. (9:97).

The Importance of Content-Addressable Memory

The importance of storing and retrieving data in parallel has been known for about 30 years (2:51). CAMs have been implemented in silicon as far back as 1966, but it hasn't been until recently, with advancements in VLSI technology, that any useful amount of CAM has been feasible (20:1003). Content-addressable memory, or associative memory, allows for this parallel access of data. CAMs can perform *read*, *write*, and *search* operations in parallel, thus substantially increasing the speed of data access.

A search operation is performed when a search pattern is sent to each cell of the CAM array. Each cell performs a comparison with this data and a match or mismatch is then signaled on the tag lines. In order to detect a match or mismatch on selected cells, it is necessary to temporarily "disconnect" cells from the tag line. This is known as "masking" off particular cells to search only selected cells. Those cells that are masked off do not affect the outcome of a search. (10:166)

A write operation writes data to cells of the CAM array. A unique feature of a CAM is that it is possible to write to all CAM words simultaneously. It is also possible to select any bit-column to perform a write operation. In doing so, only the selected cells are written to while the rest are masked off and unaltered. (10:166)

A read operation retrieves the contents of cells in a CAM array. Again, a certain bit-column can be masked so a read will not be performed in the cells of that column but only in the selected cells. (10:166)

The unique approach of memory access in CAMs can be used to increase efficiency in many application areas. Some examples are databases, pattern recognition, data correlation, speech recognition, spelling checking, language translation, neural networks, and data retrieval.

Disadvantages and Advantages of CAMs

Disadvantages. Hanlon (6:519) points out that there are surprisingly few disadvantages found in the literature on content-addressable and associative memory systems. Chisvin (2:54) states that there are a number of obstacles to overcome before commercially successful associative memories are available. These obstacles are:

- functional and design complexity of the associative subsystem,
- relatively high cost for reasonable storage capacity,
- poor storage density compared to conventional memory,
- slow access time due to available methods of implementation, and
- a lack of software to properly use the associative power of the new memory systems.

Advantages. Although these disadvantages exist, advantages to CAMs abound. When searching a content-addressable memory for data, the time to access the data is independent of the size of the CAM; all searching is done is parallel. Sorting is unnecessary because the data can easily be found by its content. Using conventional memory, the time to perform a search and sort grows at a rate of $O(n\log n)$, where n is the number of items on the list. If only the maximum value is needed, the time to find it would increase

at least as fast as the size of the list. With content-addressable memories, the time to find the maximum value would be the same despite the length of the list. (2:54)

Another advantage is the "minimization or total elimination of many of the burdensome bookkeeping operations connected with the use of conventionally organized memories". CAMs would make this bookkeeping a much easier task on programmers. It could even reduce the operating time of functions that use slower I/O devices. (6:510)

A paper written by P. M. Davies and described by Hanlon (6:518) states that processing time for many operations can be reduced "because

a) it is not necessary to store data in sorted order;

b) lookups can be made on the basis of different keys at different times over the same data;

c) records need to be stored only once;

d) addresses are not needed to store records."

Organization of list structures is accomplished very quickly with a CAM system. On the other hand, RAMs use more execution time in forming the lists, searching the list, retrieving data from a large list, deleting a list, and transferring a list to another storage medium. These functions, which require more RAM access time, are performed by CAMs in constant time. (6:518)

The repetitive structure of a CAM array makes for ease of fabrication and testing. This is also true for RAM. The CAM cell can be laid out in an organized fashion; therefore, the interconnections are short and easily implemented in integrated circuit technology. (6:518)

CAMs can also be viewed as fault tolerant. If a cell fails, that cell can be masked off, never to be used again. Of course, fault tolerance is dependent on the application of the CAM. If the system can be fault tolerant, the maintenance task is decreased. (6:518)

The Organization of CAMs

CAMs have the basic organization depicted in Figure 2. Minor variations do exist but the general concept of the interfaces remains the same. The biggest difference lies merely in terminology. The layout of a CAM array is shown in Figure 3 and relates directly to Figure 2 The definitions below contain various terms (in parentheses) used to describe each component (the list of terms is not complete).

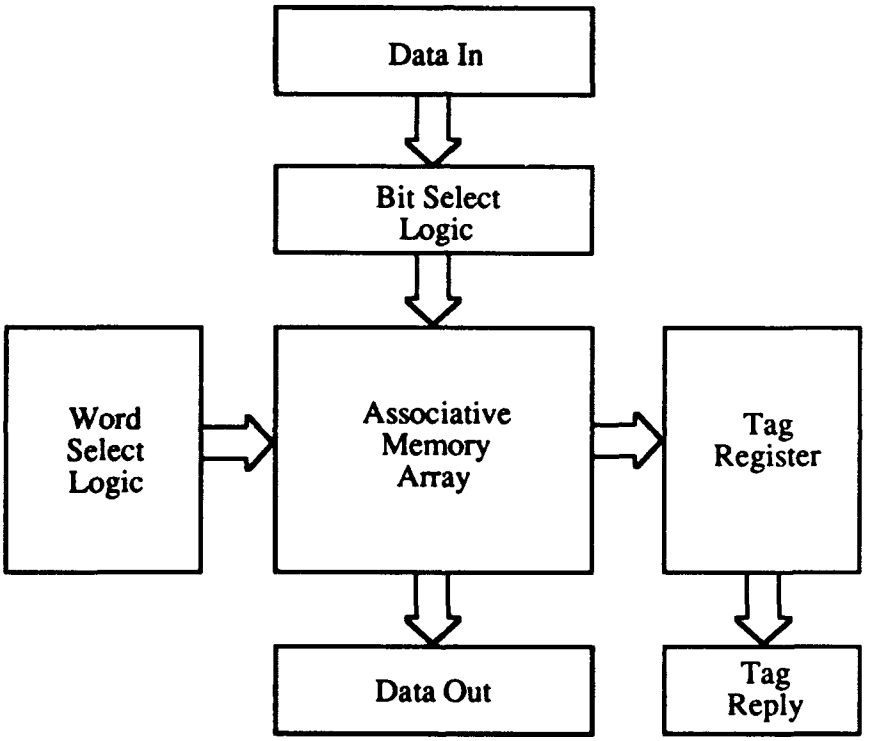


Figure 2. Organization of CAMs (3:85)

Data In (Argument Register, Data Input Register). These registers contain the data to be read or searched for as well as the data that will be written into the array. Figure 3 displays this as the Data Input Register. (3:84)

Bit Select Logic (Mask, Mask Register). This register is used to specify the bits of a word to be written to and searched for. A '1' in the bit select stream means that the

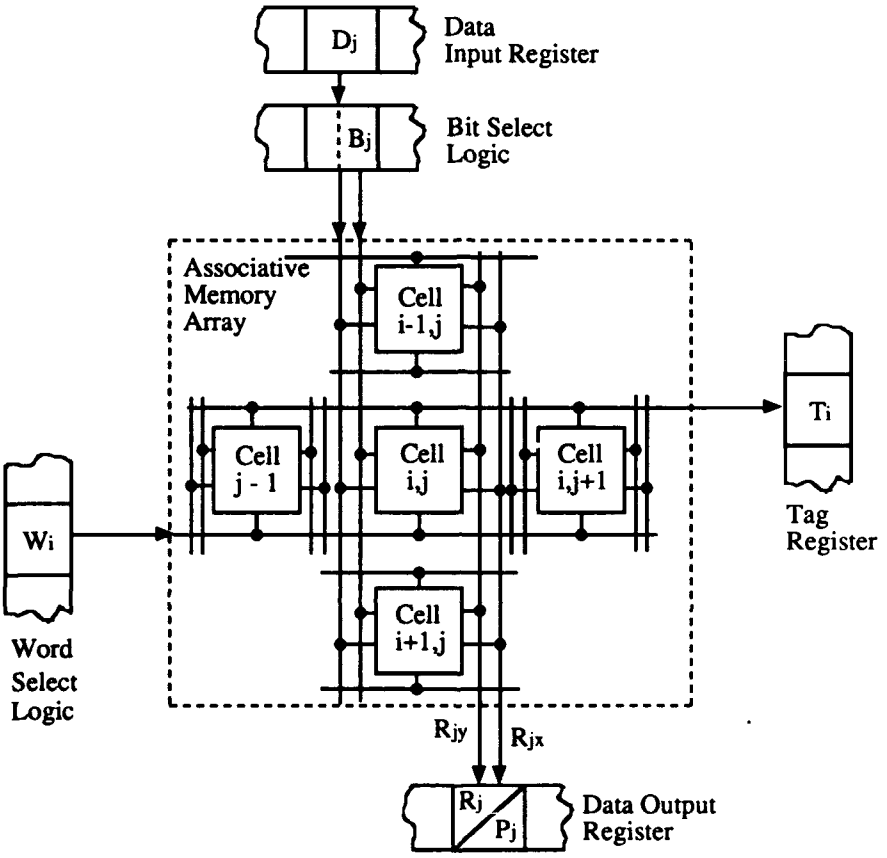


Figure 3. Associative Memory Array (3:83)

bit will be written to during a *write* operation. For a *search* operation, a '1' in the bit select stream means that a match is needed for that bit, while a '0' indicates a match is not needed (i.e., the bit is masked off). (3:84)

Word Select Logic (Decoder, Address Decoder). This register indicates which words are to be used in a *read* or *write* operation. (3:84)

Associative Memory Array (Memory Cells, CAM Array, Memory Array). This is the memory portion of the organization that contains CAM cells laid out in a 2-dimensional array, as shown in Figure 3.

Tag Register (Response Store). This register indicates that a selected word has matched the portions of the data input specified by the bit select register. (3:84)

Data Out (Output, Data Output, Data Output Registers). This register stores the data of the selected word from a *read* operation. The Data Output Register of Figure 3 contains the data that is output from the array as well as a "valid" field. The R_j field is the data while the P_j contains the valid bits. These are discussed in more detail in Chapter 3. (3:86)

Tag Reply. This contains a reply to the control unit that one or more set tags are in the tag register. (3:85)

CAM Data Word Arrangement

A common CAM data word arrangement is shown in Figure 4. The *Tags* field shows the type of data stored at that location (i.e., data or code) and whether the location is empty or used. The *Label* field is used in the comparison operation. Finally, the *Data* field is the storage area for the information to be retrieved or modified. Sometimes the *Label* and *Data* fields are treated as one if the *Label* field is part of the *Data*. (2:52)

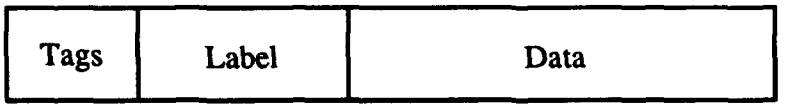


Figure 4. CAM Data Word Arrangement (2:53)

The data word arrangement is flexible. The data word could be segmented in any way with any of the segments used for interrogation. Or it may not be segmented at all, in which case any choice of bits can be selected for an interrogation (this is the most general form of a CAM). (6:509)

Rowe (18:15) points out that various data-word sizes have been used by different people and organizations for diverse reasons. They range from 30 to 140 bits in length.

A 30-bit wide data-word does seem a bit strange but one reason cited for it was that it matched the CPU's word length.

Types of Architectures

Four different types of architectures exist for associative memories: bit-serial, byte-serial, word-serial, and distributed logic. The tradeoffs among these types of memories consist of the storage media, the communications between the cells, the type of retrieval logic, and the nature and size of external logic (such as registers and I/O ports). (2:58)

Bit-serial associative memories search the data a bit at a time, in parallel, in each word of the associative array. The search time, therefore, depends on the word width and is independent of the number of words, or word depth, of the array. After one bit-slice is searched, the next bit position in each word is inspected, and so on, until the entire field is searched. Likewise, byte-serial and word-serial associative memories search the data a byte and word at a time, respectively, in parallel for the depth of the memory.

Distributed-logic memories avoid the serial aspects of the bit-, byte-, and word-serial memories by placing the search, read, and write logic into each cell. This allows all memory cells to be accessed simultaneously in parallel. Since the logic is in each cell of the memory, the cell cycle time is longer in the distributed logic array than in the bit-serial architectures. (2:59)

Design considerations must be taken into account when deciding upon which architecture to choose. The bit-and byte-serial architectures work best on data and arithmetic computations. Distributed logic arrays work best on equality comparisons and multiprocessor control. Cost is another factor. The amount of logic in the distributed memory cell causes this type of memory to be physically larger and more expensive than

the bit- or byte-serial memories. The lack of cell logic in the bit- or byte-serial architectures allows these memories to be denser than the distributed logic arrays. (2:59)

Current CAM Cell Designs

Several CAM cell designs are available in the literature. The hardware implementation of these designs uses one of the following three options:

1) Static. Data are stored using two cross-coupled inverters acting as a flip-flop. The data remains in the cell as long as power is supplied. When power is taken away, the contents of the cell are lost.
2) Self-refreshing or pseudo-static. Data are stored by making use of the capacitance on the transistor gates. Over time, the charge will decay, but by asserting a control signal the charge can be restored. The consequence of this operation is increased power consumption.
3) Dynamic. Data are stored in much the same way as in RAM memory and requires external control logic to sense and refresh the data signals. The logic needed and the requirement to regularly refresh these cells results in a time penalty that may significantly slow down its operations. (10:167)

Gate Level CAM Designs. DeCegama (3:87) proposed the gate level design shown in Figure 5. A CAM chip can be modeled by collecting these cells into a 2-dimensional array, as shown in Figure 3.

Another CAM cell design is given by Hayes (7:452), and is shown in Figure 6. This design uses a D-type flip-flop for storing the data. Its match circuitry is composed of an exclusive-NOR gate. Other circuitry is present for the reading and writing functions.

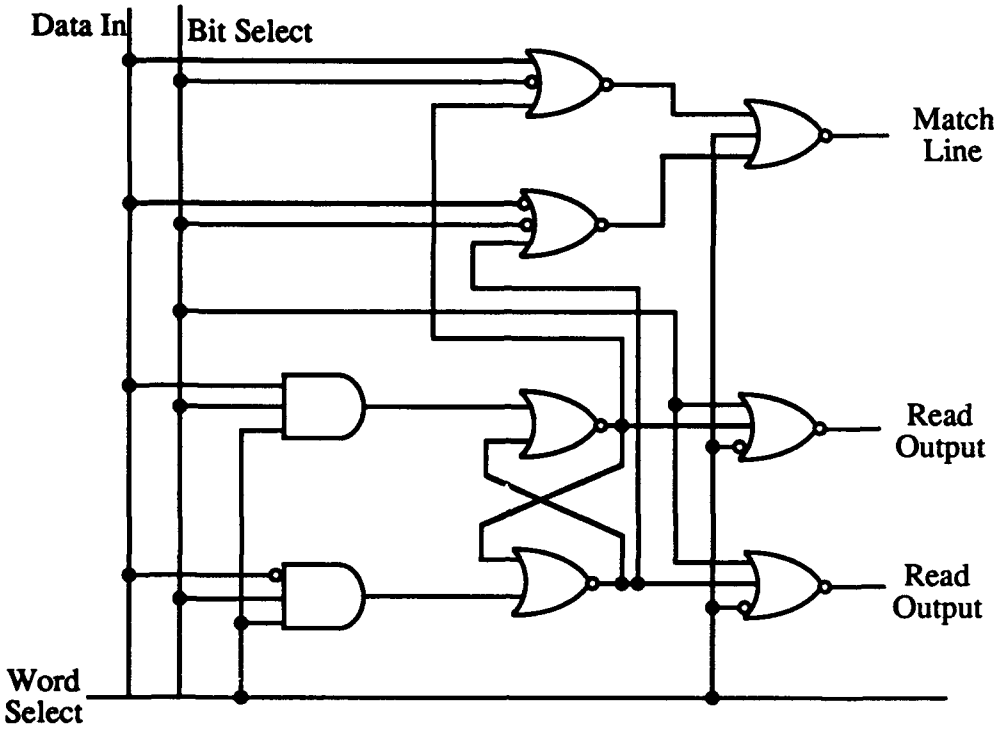


Figure 5. Gate Level Design of CAM by DeCegama (3:87)

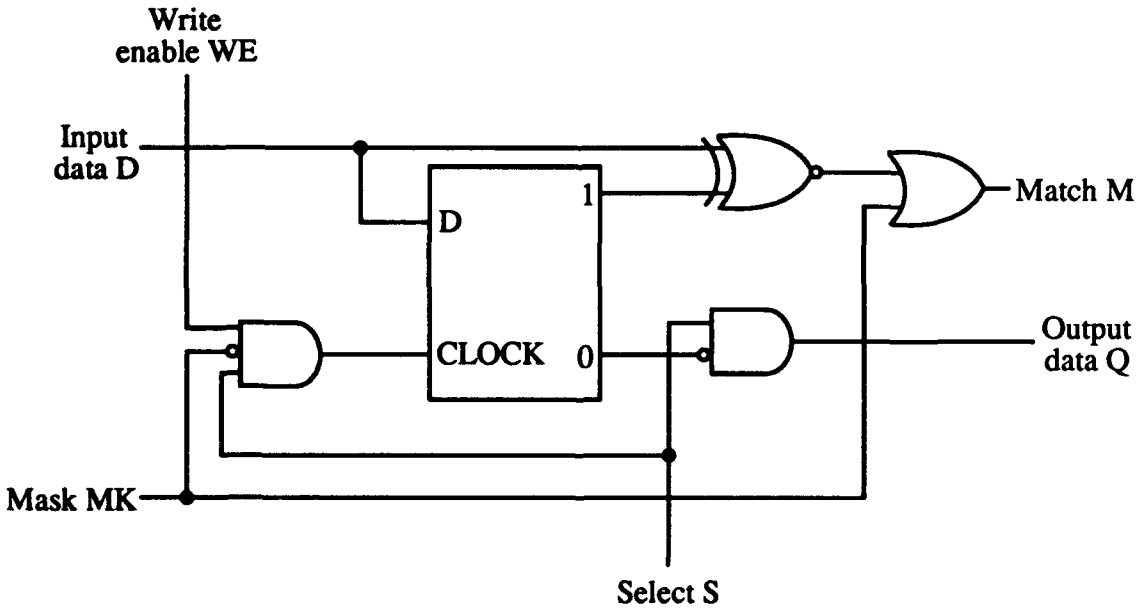


Figure 6. Gate Level Design of CAM by Hayes (7:452)

Transistor Level CAM Designs. The CAM cells described above were designed at the gate level. Several more cells can be found in the literature that are designed at the transistor level. One such design is the AFIT CAM designed by Shinn (19:13), shown in Figure 7. This is a general purpose CAM that can search the cell contents on the basis of equality, between limits, greater than, less than, etc.

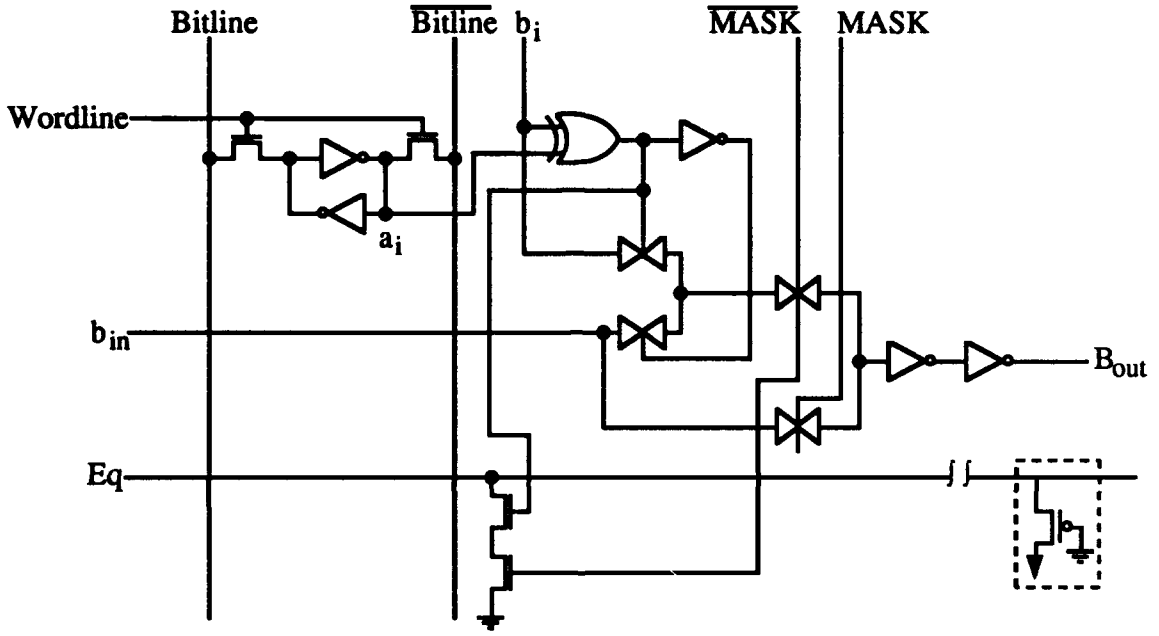


Figure 7. Gate Level Design of CAM by Shinn (19:50)

Another transistor level design is shown in Figure 8. This CAM cell was developed for large-bit-capacity CAM LSI to realize a partial-WRITE operation. "The associative-memory cell circuit is composed of seven/nine n-MOS transistors and two high-resistive poly-Si load devices." (16:1014)

Figure 9 shows a five-transistor dynamic CAM cell. This cell can store three states: ZERO, ONE, and the DON'T CARE state. "A cell in the DON'T CARE state is unable to discharge the match line." (20:1006)

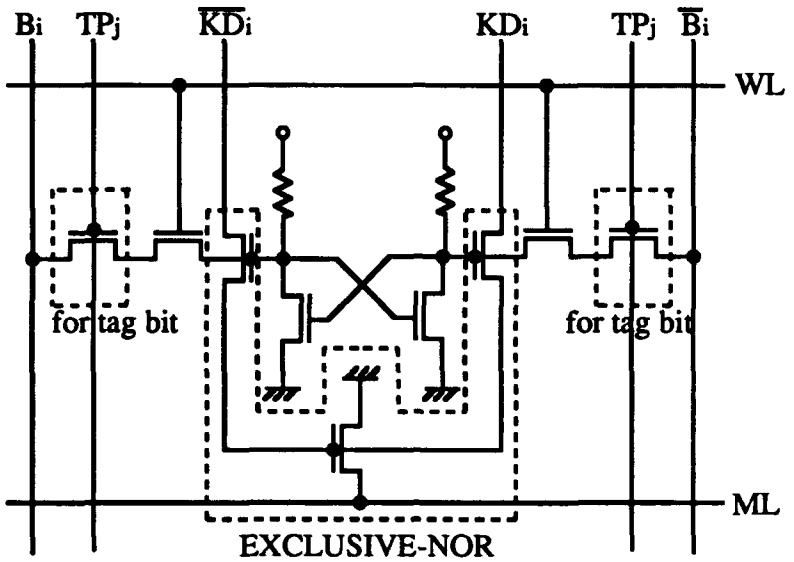


Figure 8. Transistor Level Design of CAM by Ogura (16:1014)

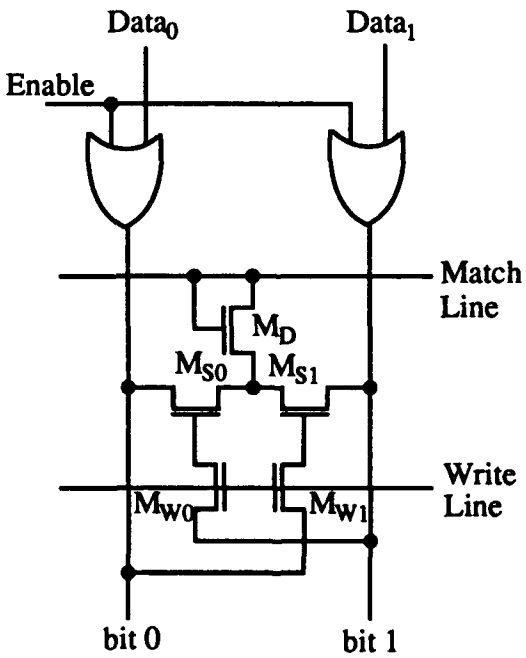


Figure 9. Transistor Level Design of CAM by Wade (20:1006)

A slightly different design than the one in Figure 9 is shown in Figure 10. This content-addressable parallel processor (CAPP) uses only 5 transistors. It has three states, with the DON'T CARE state "being useful in logical inferencing and pattern-matching applications." (8:537)

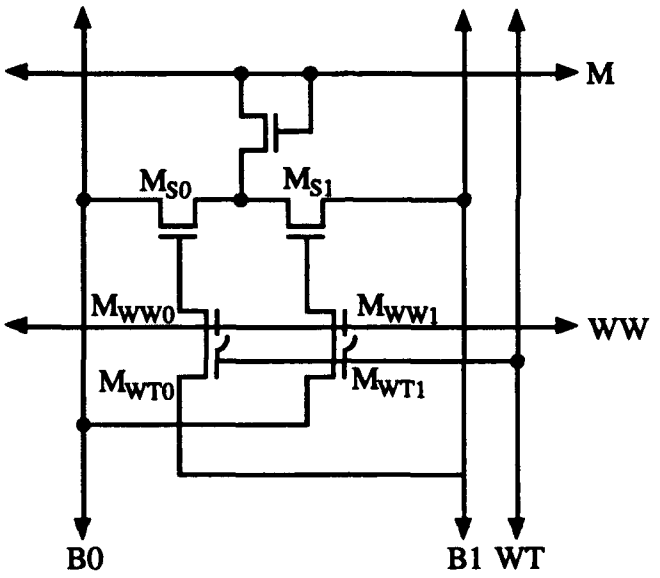


Figure 10. Transistor Level Design of CAM by Herrmann (8:538)

A 12-transistor CAM cell is shown in Figure 11. This is an addressable CAM that functions as a normal CAM even though its data store is RAM-based. It is made up of three sections: 1) a six-transistor static RAM, 2) four-transistor XOR, and 3) a two transistor parallel write pull-up disable gate (DISABLE). "It was designed primarily for address translation in a high-speed packet switching network." (13:257, 258)

Weste (21:351) briefly describes a transistor-level CAM cell designed by J. C. L. Hou. This cell nine-transistor cell is shown in Figure 12.

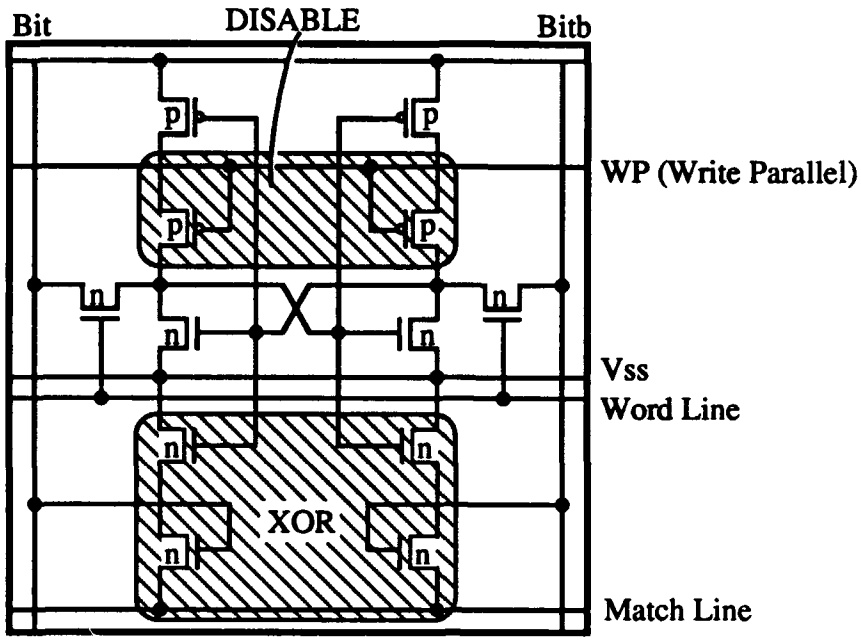


Figure 11. Transistor Level Design of CAM by McAuley (13:258)

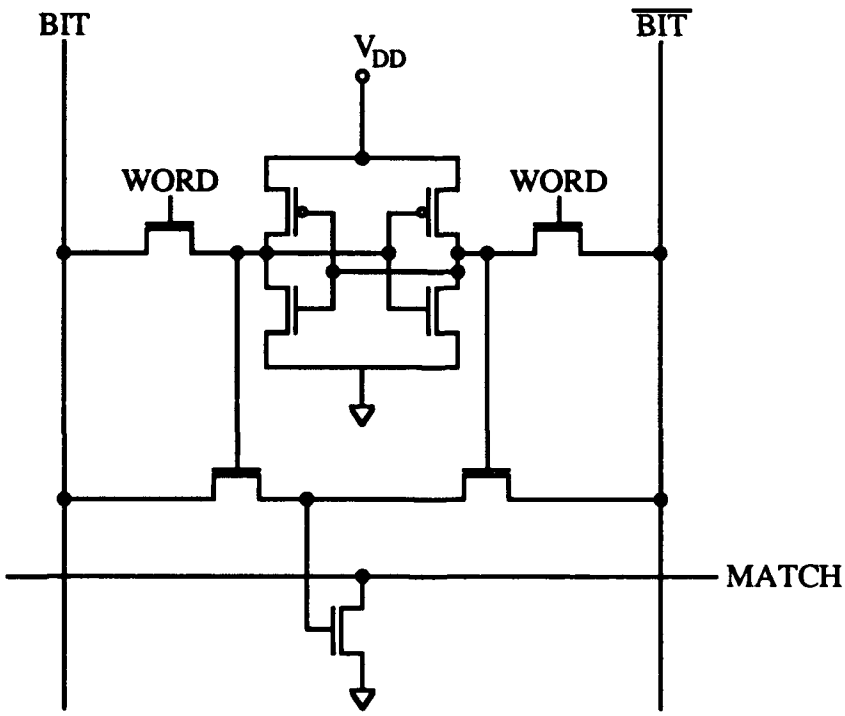


Figure 12. Transistor Level Design by Weste (21:351)

Finally, Jones (10) presents four different transistor-level CAM designs. Jones discusses the design constraints, trade-offs, and implementation issues involved in deciding which design to choose for a VLSI CMOS high-speed CAM architecture. The designs are shown in Figures 13 through 16. (10)

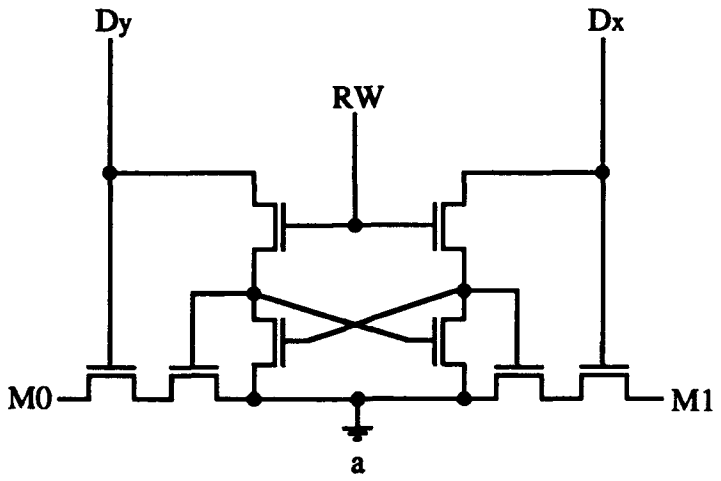


Figure 13. CAM A Transistor Level Design by Jones (10:167)

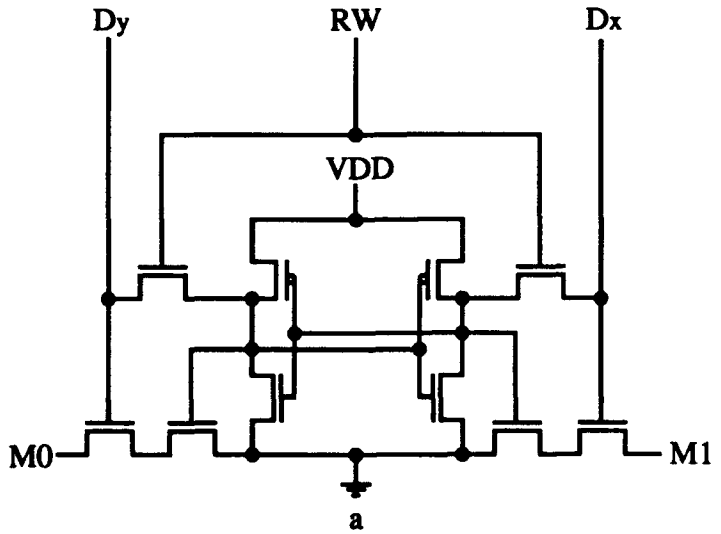


Figure 14. CAM B Transistor Level Design by Jones (10:168)

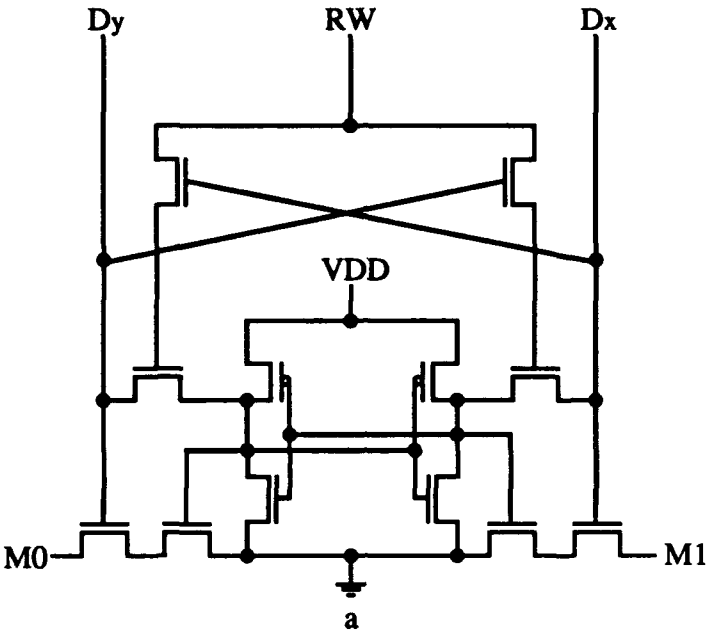


Figure 15. CAM C Transistor Level Design by Jones (10:169)

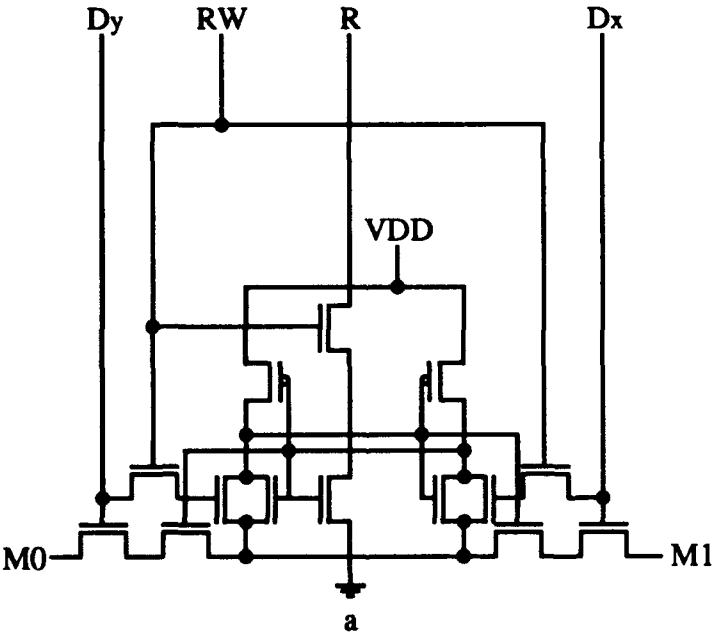


Figure 16. CAM D Transistor Level Design by Jones (10:169)

Figure 13 shows the simplest design considered by Jones. The cell is "pseudo-static in that when the *RW* line is driven high, the cell acts as a static nMOS flip-flop" (10:168). This cell needs to be refreshed frequently so data will not be lost. (10)

Figure 14 is very similar to Figure 13 but the data are stored in two cross-coupled CMOS inverters. This design does not require refreshing. (10)

Figure 15 is an enhancement of Figure 14. This design "avoids the need to rely on the positive feedback between the cross-coupled inverters to maintain the data contents of the CAM cell..." (10:168). This design reduces power consumption on the 'write X' operation. (10)

Figure 16 is different from Figures 13 - 15 in that it has 'capacitive' loading on the data lines. "The design relies on the positive feedback between the two cross-coupled inverters to complete the write operation" (10:168). In contrast to Figure 15, both drive transistors are switched off during a 'write X' operation, which reduces power consumption. (10)

Jones chose CAM A based on power consumption and chip area. Jones found "the 'search' and 'read' operations, being based on a precharge/discharge mechanism, consume relatively little power. It is during the 'write' operations that the designs exhibit different, and often quite large, power consumption figures" (10:169,170). Table 1 shows that CAMs B to D (the CMOS designs) have a much higher power consumption during the 'write 0 and 'write 1' operations than CAM A. Conversely, CAMs C and D consume very little power during a 'write X' operation compared to CAM A, with CAM B consuming the most.

An interesting result occurs when these CAMs are introduced into their working environment. Table 2 reveals that CAM A has the highest power consumption. "This is caused by the predominance of 'write X' operations, and the need to refresh the memory

Table 1

CAMs A through D Power Consumption in Isolation (10:169)

	CAM area (μm^2)	Write 0/1 (μW)	Write X (μW)
CAM A	1944	51.0	99.8
CAM B	2980	149.0	120.6
CAM C	4160	165.4	<1
CAM D	5240	173.8	<1

Table 2

CAMs A through D Average Power Consumption in Working Environment (10:170)

Name	Power (μW)
CAM A	64.7
CAM B	53.6
CAM C	10.6
CAM D	11.2

at regular intervals" (10:170). Relative to CAM A, CAMs C and D consume little power since the 'write 0' and 'write 1' operations are performed infrequently. Figure 17 shows the chip power dissipation of each CAM design. (10:170, 171)

The other consideration in the selection ot CAM A was the chip area used by each design. Figure 18 shows the chip area needed for each CAM. Notice that the more sophisticated CAMs (C and D) require the largest die size. The smallest area required by the four CAM designs came from CAM A, which was a major factor in its selection. (10:171)

Summary

The goal of content-addressable memories is to access data in parallel based on content rather than by address. The memory subsystem of Figure 1 will use CAMs to increase the speed of memory accesses over that of conventional hierarchies. A brief

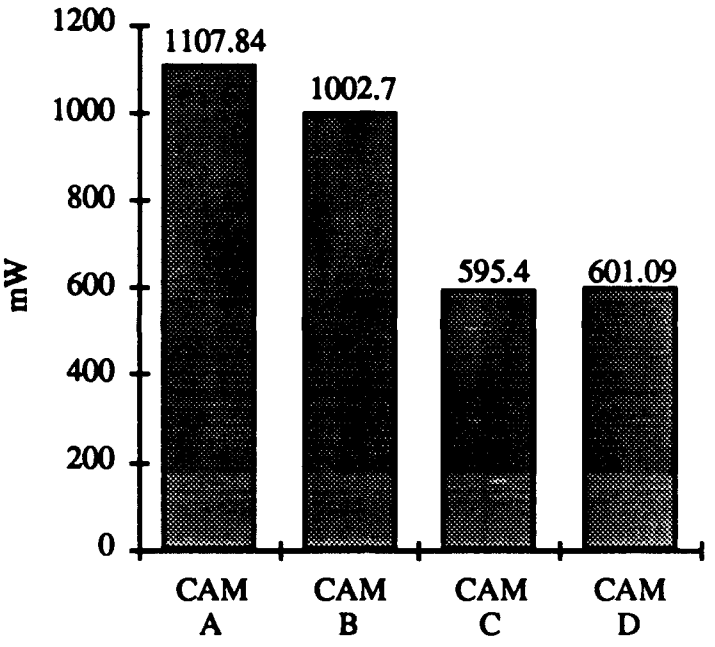


Figure 17. CAMs A through D Power Dissipation (10:171)

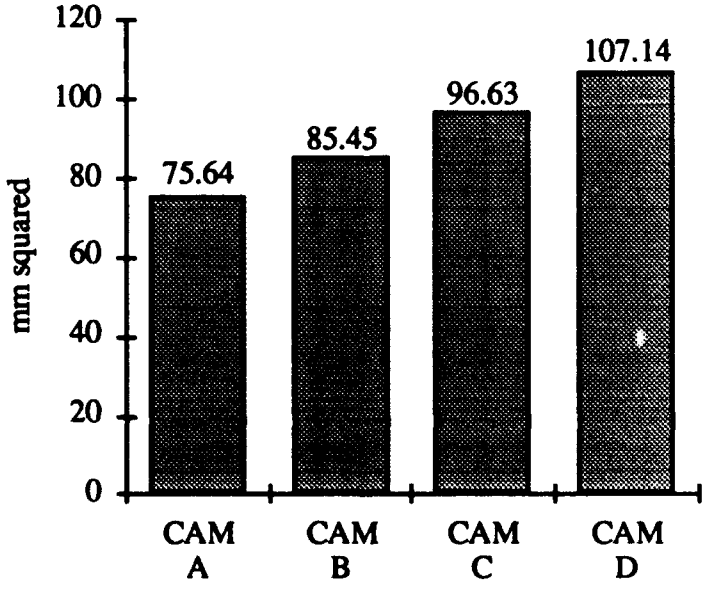


Figure 18. Chip Area Needed for CAMs A through D (10:171)

overview of the organization of CAM chips was presented. Several different CAM implementations were shown, none of which has been previously used in a two-level cache memory architecture.

III. The Design of the Main CAM Cache that Exploits Structural Locality

Overview

This chapter covers the design of the main CAM cache (MCC). The MCC performs two basic functions: *read* and *write*. Upon activating the MCC to perform these functions, a search operation is performed on the address that is made available to the MCC. After the search is complete, one of four distinct states exists: *Read Hit, Read Miss, Write Hit, and Write Miss.*

The Read Hit state is the most important state the MCC can be in. It is in this state that the prefetching of structural locality of memory references is performed. The state is entered when the CPU requests that the MCC perform a read on the requested address. If the address is stored on the MCC, a read hit (i.e., the search operation found the address) occurs. This triggers the prefetching of data in a first-in-first-out (FIFO) manner.

The Read Miss state is also entered during a read cycle. First, a search is performed on the CAM array. If the search is unsuccessful, the Read Miss state is entered. Since the address was not found (i.e., "missed"), the FIFO replacement algorithm is used to write the data into the cache. These data come from main memory over the data bus. Thus, temporal locality is captured by the MCC.

The Write Hit state is entered during a write cycle. Again, the first operation performed by the MCC is the search operation. If the address used in the search was found, then that location in memory will be replaced by the new data.

The Write Miss was the easiest state to deal with. If the search operation produced no matches, the MCC does nothing but wait until the next operation is requested of it.

The remainder of this chapter describes the VHDL implementation of the MCC. Since a bottom-up approach was used to design the cache, it is appropriate to describe the MCC in a bottom-up fashion. First, the method used to design the MCC is briefly described. Then, a brief look at the basic components that make up the MCC are discussed. Next, the heart of the cache, the CAM cell, and the inner workings of the CAM array are described in detail. The brains of the MCC, the controller, is then explained. Finally, the MCC is viewed as a whole and its functionality is presented.

Design Methodology

The first step taken in this thesis effort was to choose a logic design of a CAM cell. The CAM cell designed by DeCegama (3:87) was used. After implementing the CAM cell in VHDL, the CAM array was built using the VHDL *generate* function.

Once the CAM array was tested thoroughly, the controller was designed. The four states described above were used to logically decide upon the components, gates, and signals that must be used in order for the MCC to function properly. A schematic diagram was drawn of the entire cache and it was broken into two major sections. The first was the CAM array and its associated logic and the second was the controller. The controller was further broken out into its major components so it could be built in a hierarchical manner. This allowed for the separate sections to be tested before being integrated into the controller portion of the MCC.

The next step taken was the building, in VHDL, of all the basic components needed in the MCC. These were the building blocks for the major components that make up the controller. Once these basic components were built, the major components of the controller section were created from them. The controller was then put together from these sections and the final step was the integration of the controller onto the MCC with the CAM array and its logic.

Basic Components

This section contains the basic components that make up the major components of the controller. The VHDL code for each of the basic components is in Appendix B. They are organized in alphabetical order.

BINARY_COUNTER: This synchronous binary counter is a slightly modified version of the binary counter in Mano (12:278). It is the only reason the MCC requires a dual-phased clock. The version shown in Figure 19 uses JK-type flip-flops with RESET (these JK-type flip-flops are described later in this section). It is also of generic size. The size, Bits_In_Counter, is defined in chip_pkg.vhd (see Appendix E).

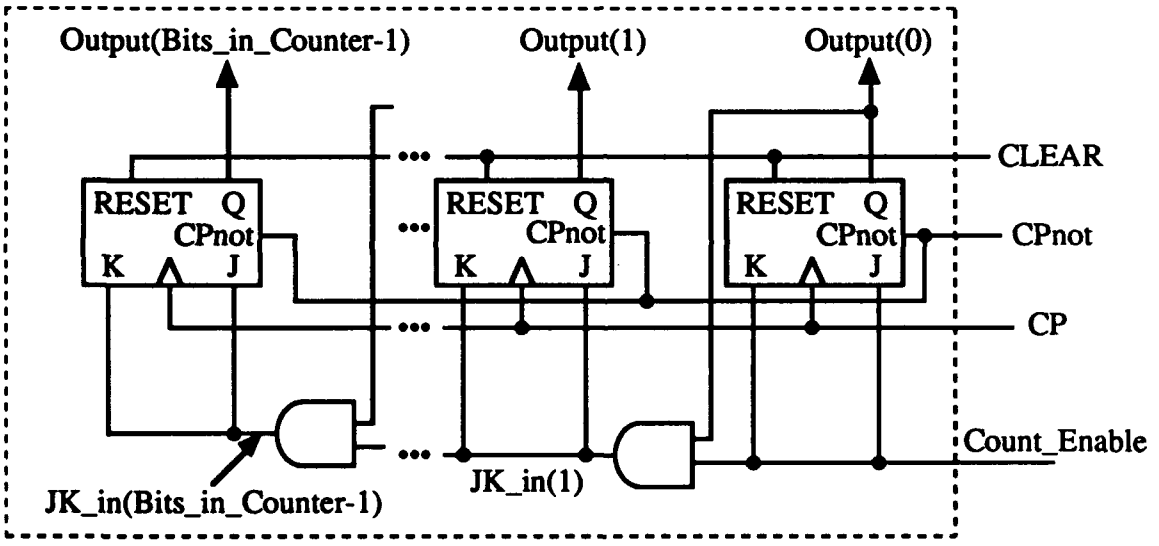


Figure 19. Schematic Diagram of BINARY_COUNTER (12:278)

CHANGE_DETECTOR (14): The purpose of this circuit is to detect a change from '0' to '1' and from '1' to '0' in any input signal. The circuitry is very simple and is shown in Figure 20. Refer to Figure 21 during the following explanation of how it works. Suppose a '0' is on both inputs of the XOR gate. This causes the output to be '0' (0 XOR 0 = 0). Now, if a '1' is input into the change detector, a '1' is on one of the XOR inputs.

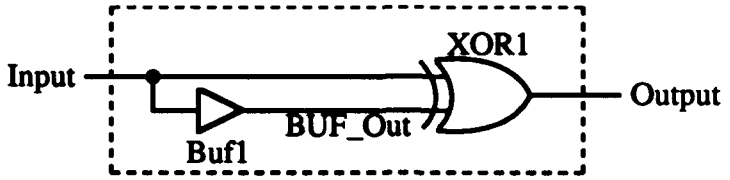


Figure 20. Schematic Diagram of CHANGE_DETECTOR (14)

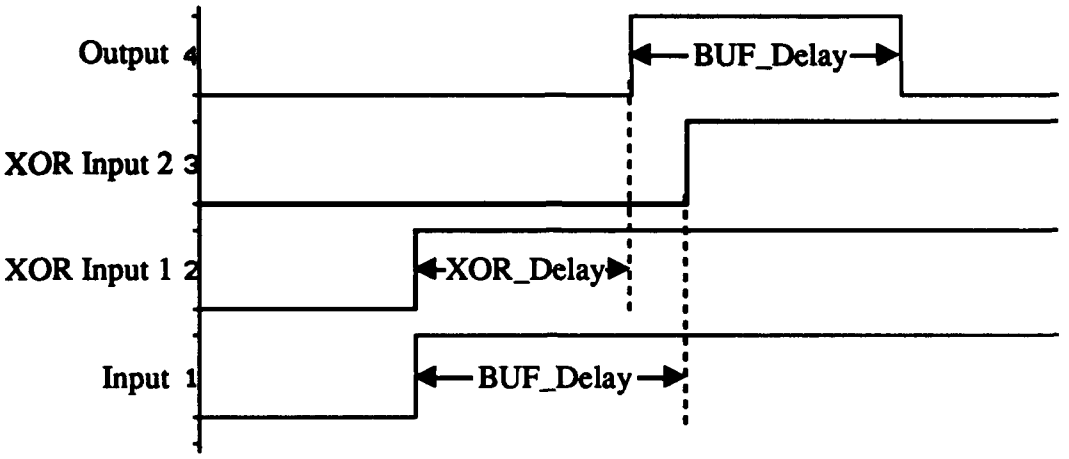


Figure 21. Timing Diagram for CHANGE_DETECTOR Operation

The other XOR input is still a '0' for as long as it takes for the signal to go through the buffer (BUF_Delay in the figure). This causes the XOR output to be a '1' (1 XOR 0 = 1). After the buffer delay, the second XOR input gets '1' and the XOR output becomes a '0' again (1 XOR 1 = 0). The circuit works the same way with a signal transition from '1' to '0'.

EDGE_TRIGGERED_DFF: This is a D-type positive-edge triggered flip-flop taken from Mano (12:214) and slightly modified. The logic diagram is shown in Figure 22. The only change was the addition of a RESET port to allow the flip-flop to be asynchronously reset to '0'.

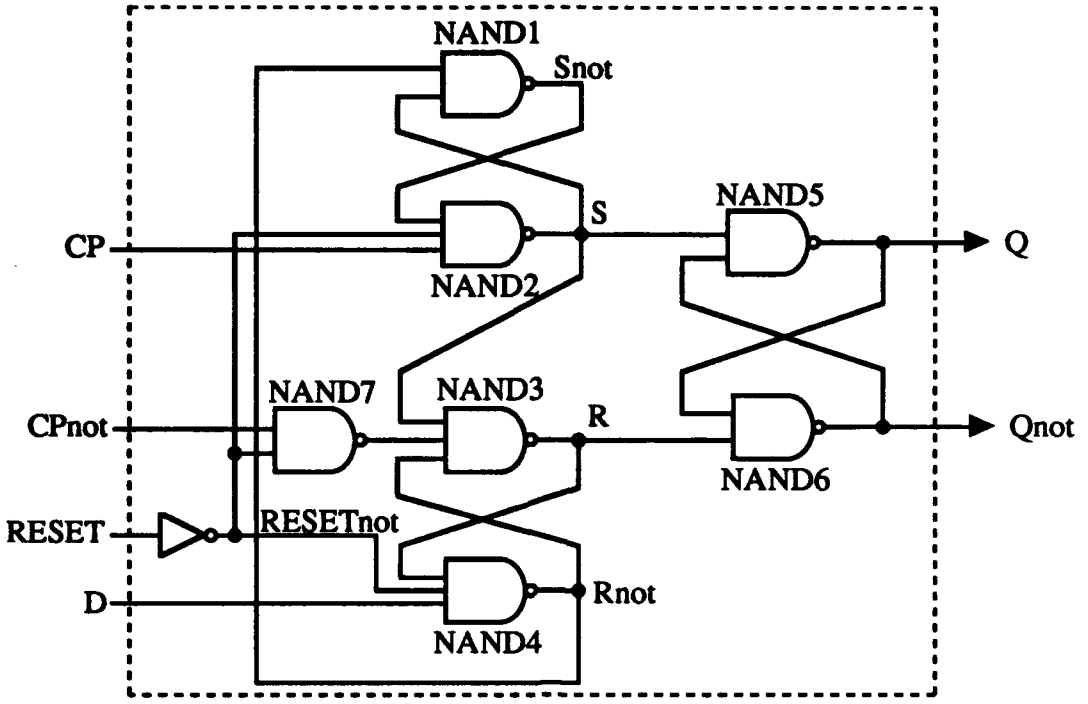


Figure 22. Schematic Diagram of EDGE_TRIGGERED_DFF (12:214)

MS_JKFF: This is a clocked master-slave JK-type flip-flop taken from Mano (12:213) and slightly modified. The schematic diagram of this component is shown in Figure 23. A RESET port was added to allow it to be asynchronously reset to '0'. In addition, the CP port was modified to allow a dual-phased clock to be used and, as a result, a NAND gate was deleted from Mano's design.

PREFETCH_COUNTER: This component is very similar to the word-time signal generator found in Mano (12:285) and is shown in Figure 24. It uses the BINARY_COUNTER of Figure 19 to count upward from zero to Prefetch_Block_Size-1 (as defined in chip_pkg.vhd of Appendix E). When the circuit is "started", the output signal of the RS-type flip-flop, Counting, produces a '1' and the binary counter begins to count. The XNOR gates are used to compare the outputs of the binary counter and Prefetch_Register. When they are equal, the output of each XNOR gate becomes '1' and

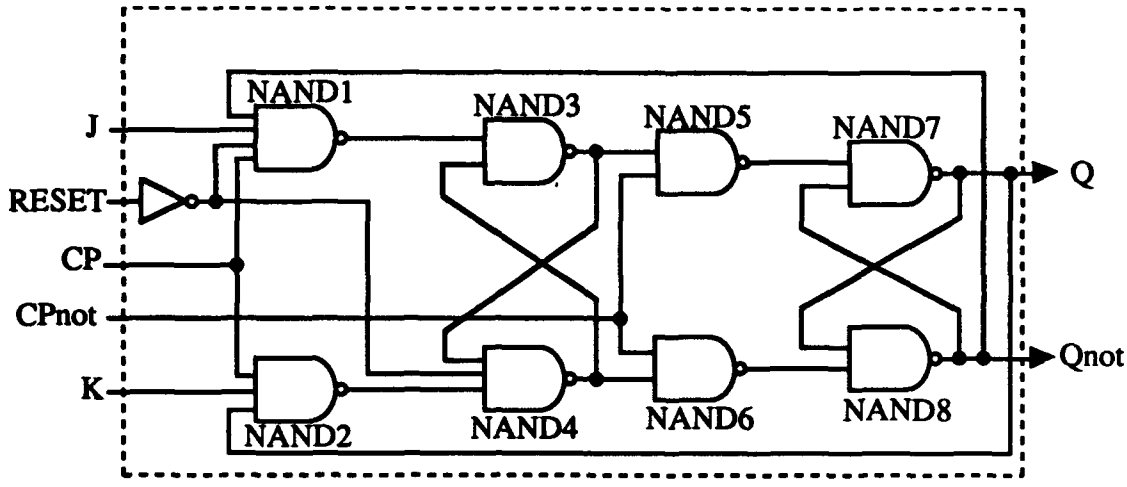


Figure 23. Schematic Diagram of MS_JKFF (12:213)

the RS-type flip-flop is reset to zero. Counting then becomes '0' and its complement is able to clear the binary counter to set up for the next prefetch cycle.

RS_FLIPFLOP: This is an RS-type flip-flop found in Mano (12:206). The logic diagram is shown in Figure 25.

SHIFT_REGISTER: This component is a circular shift register with parallel load. The design came from Mano (12:267) and was implemented with some modifications. This register is a unidirectional shift register, whereas Mano's is bidirectional. Therefore, 2x1 multiplexers were used as opposed to Mano's 4x1 multiplexers. Also, SHIFT_REGISTER is of generic size whose depth is defined in the chip_pkg.vhd of Appendix E. SHIFT_REGISTER is shown in Figure 26.

TOS_SHIFTER: This shift register is very similar to that of Mano's shift register (12:264). It uses the EDGE_TRIGGERED_DFF with RESET and is shown in Figure 27. It is cleared upon initialization of the MCC with Master_Reset and loads the bottom flip-flop with a '1'. The Master_Reset signal is connected to the CLEAR port of TOS_SHIFTER. When it goes high, all flip-flops are reset except for the bottom one. An OR gate is connected to the D input of this flip-flop and the OR inputs are the topmost

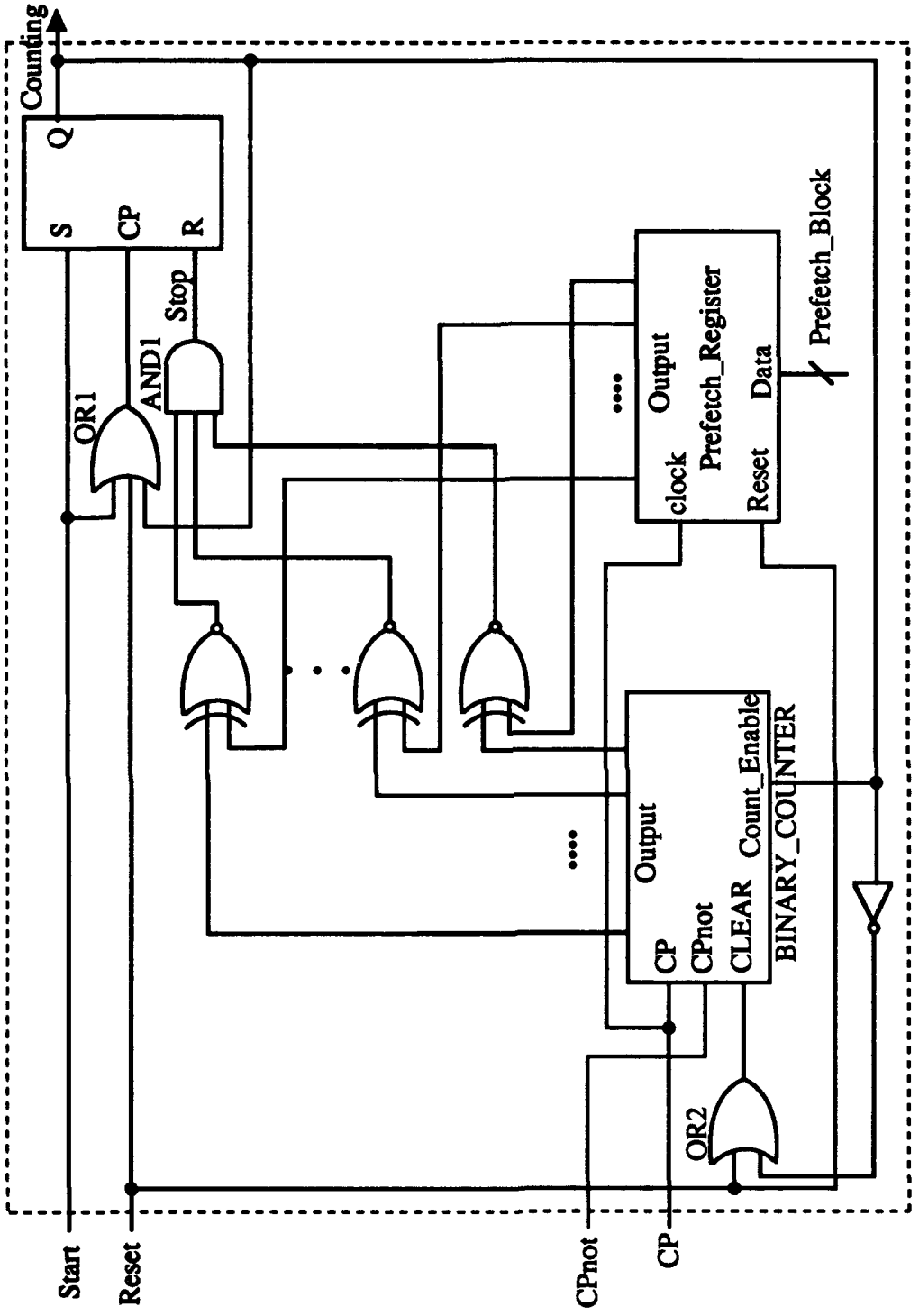


Figure 24. Schematic Diagram of PREFETCH_COUNTER

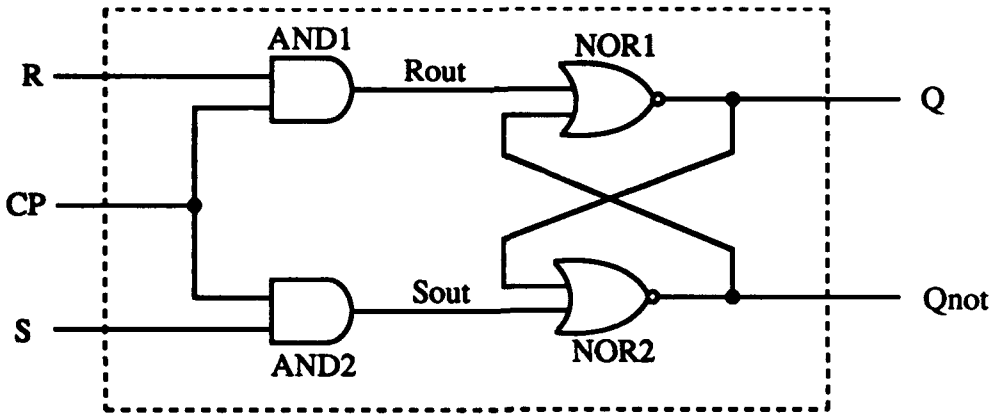


Figure 25. Schematic Diagram of RS_FLIPFLOP (12:206)

flip-flop's output and the CLEAR port. Therefore, the circular shift can be accomplished with the top flip-flop's output, and the CLEAR port can load the bottom flip-flop with a '1' when Master_Reset is '1'. The CLEAR port is also ORed with CP and input into the CP port of the bottom edge-triggered D-type flip-flop. Thus, upon initialization, the bottom flip-flop clocks in a '1', which can then be shifted in a circular manner.

The CAM Cell

The fully-associative CAM cell was selected over the bit-slice, byte-slice, and word-slice associative memories for the implementation of the structural locality cache memory subsystem. Its advantages far outweighed its disadvantages for the purpose of this research. Speed during memory accesses is the most important issue (outweighing the fact that more logic is needed to form each cell) and the fully associative array is fast. Each cell of the CAM array performs its comparison simultaneously, thus increasing the speed of the entire memory subsystem.

The CAM cell proposed by DeCegama (shown in Figure 5) was used in the implementation. It is a fully associative CAM cell with all the features necessary to be integrated onto a cache used to prefetch memory references. This design was chosen for this thesis not because of its robust functionality but simply because it was familiar to the

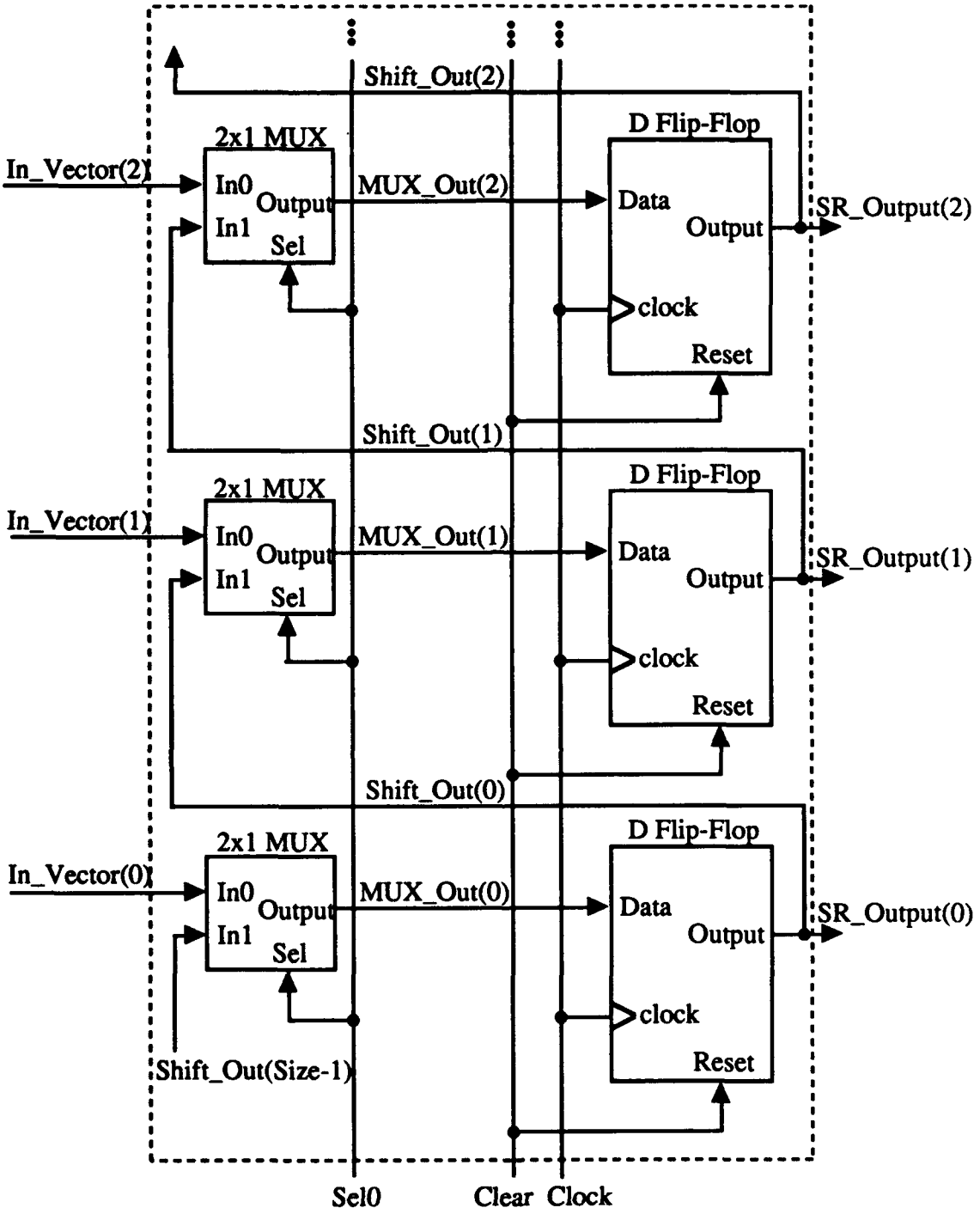


Figure 26. Schematic Diagram of SHIFT_REGISTER (12:267)

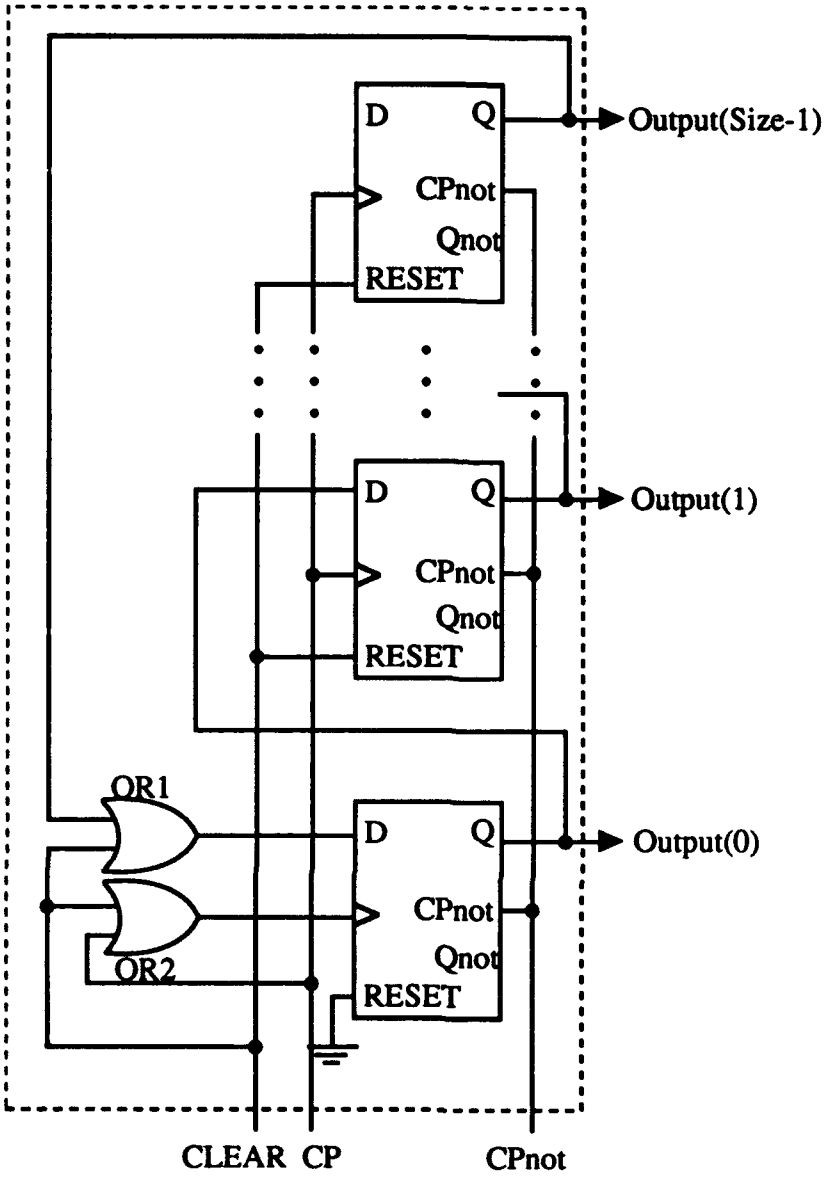


Figure 27. Schematic Diagram of TOS_SHIFTER (12:264)

author. The structural description of the cell was directly implemented into VHDL using ZYCAD™'s (22) gate components. The naming of the gates and signals is shown in Figure 28. The VHDL code for the CAM cell is shown in Appendix A.

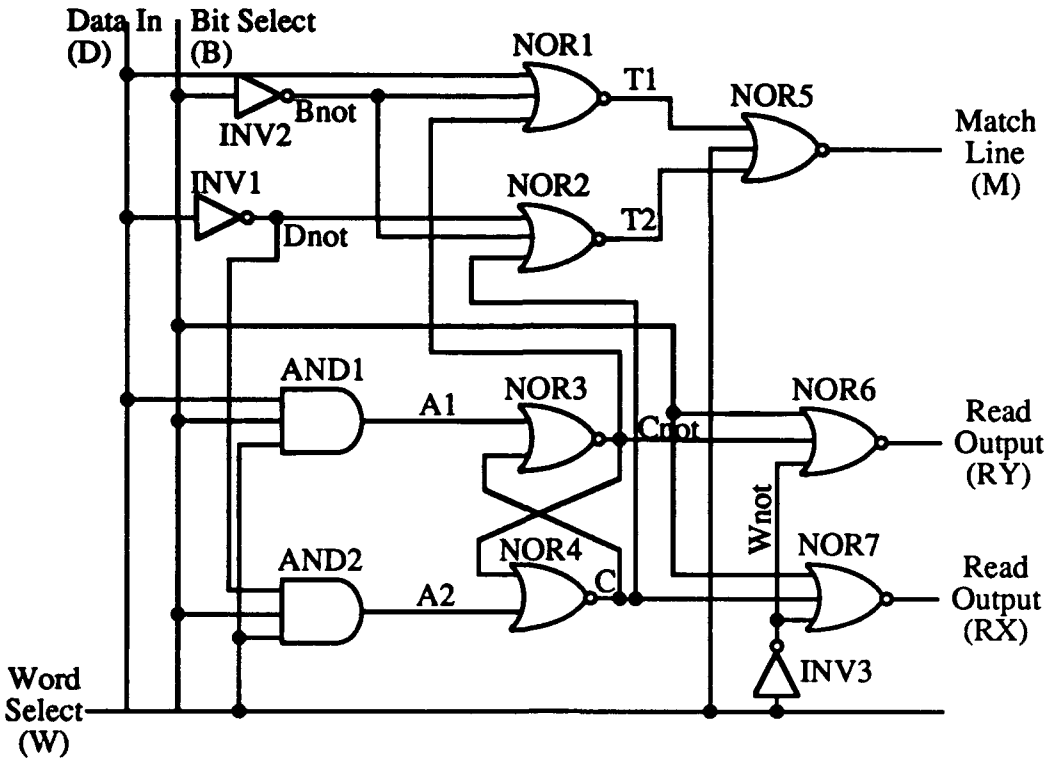


Figure 28. Naming convention for CAM cell gates and signals

The CAM cell performs three functions: *search*, *read*, and *write*. The RS-type flip-flop, consisting of gates AND1, AND2, NOR3, and NOR4, is used to store the contents (C) of the cell. Table 3 shows the inputs to the CAM cell to perform the desired operation.

Table 3

CAM Cell Inputs for Desired Operation

	Bit Select (B)	Word Select (W)
search	1	0
read	0	1
write	1	1

During a *search*, the datum to be searched for (D) is placed on the *Data In* line, as shown in Figure 28. Simultaneously, the *Bit Select* line (B) is set to '1' and the *Word Select* line (W) is set to '0'. Since B = '1', both *Read Outputs*, RX and RY, produce a '0' from gates NOR7 and NOR6, respectively. If D and C are equal, then a '1' will go into the NOR1 and NOR2 gates producing a '0' as their outputs. Since these outputs go into gate NOR5 and W = '0', then Match Line (M) becomes a '1'. Conversely, if D and C are not equal, either gate NOR1 or NOR2 will produce a '1' as its output, forcing the output of NOR5 (M) to go to '0'. Logically, M is derived as follows:

$$M = [(D + B' + C')' + (D' + B' + C)' + W]'$$

but B = 1 (or B' = 0) and W = 0, so

$$= [(C' + D)' + (C + D')']'$$
$$= [CD' + C'D]'$$
$$= (C' + D)(C + D')$$
$$M = CD + C'D'.$$

During a *read* operation, W is set to a '1' and B is set to a '0'. Note that D does not affect the result of the *read*. The output M goes to '0' from gate NOR5 since W is a '1'. The content of the cell is output through NOR6 onto RY and its complement is output through gate NOR7 onto RX. Logically, RY and RX are described as follows:

$$RY = (B + C' + W')'$$
$$= B'CW$$

but B = 0 (or B' = 1) and W = 1, so

$$RY = C$$

and

$$RX = (B + C + W')'$$
$$= B'C'W$$

or

$$RX = C'.$$

During a *write* operation, both B and W are set to '1'. Since W is a '1', M becomes a '0', and since B is a '1', RX and RY become '0'. The cell content C will become D as proven below:

$$C = [(DBW + C)' + (D'BW)]'$$

now since B and W are both '1',

$$C = [(D + C)' + D']'$$
$$= [C'D' + D']'$$
$$= [D']'$$
$$C = D.$$

After the cell was completely tested, an array of cells was organized to form the CAM array.

The CAM Array

VHDL has a useful function called *generate*. Using this feature, an *m* by *n* array of CAM cells was generated. This allows the decision about the size of the array to be delayed to a later time when details about the fabrication technology and chip size are considered. In the VHDL description of chip_pkg.vhd (Appendix E), *m* is the length of the word (*Word_length* in the code) and *n* is the depth (*Depth* in the code) of the array (see Figure 29).

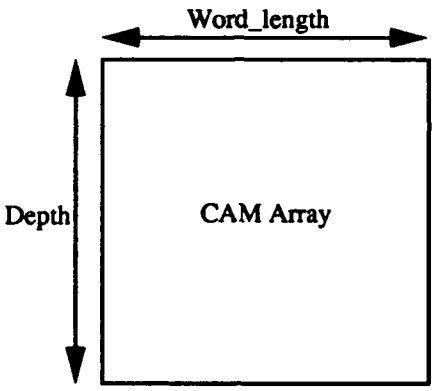


Figure 29. Dimensions of the CAM Array

The organization of the CAM array is shown in Figure 30. It functions as follows. The Address_Buffers and Data_Buffers accept incoming data from the MCC's Address_In and Data_In ports, respectively. These buffers in turn provide the buffered data to the Data_In_Bus. The buffers act to amplify the data before going into the CAM array. The Bit_Select_Bus and the Word_Select_Bus get data from the controller. This arrangement allows selected bits of a word to be compared during a search operation, and multiple words to be written to and read during the write and read operations.

The resolution functions, Wired_Or and Wired_And, are used to resolve signals along selected buses inside the CAM cache. The VHDL code for these resolution functions came from Lipsett (11:103-105) and can be found in chip_pkg_body.vhd of Appendix E. The implementation into hardware is technology dependent. The VHDL description would be modified to match the specific hardware technology used to realize the design.

When a certain bit is to be compared during a search, a '1' is placed on the Bit_Select_Bus to select that bit in all words of the array. If a match is successful, an M output of '1' from each matched cell is placed on the Resolved_Signal_Tag line. If a match is unsuccessful, a '0' is placed on the line. This line is a wired-AND that resolves

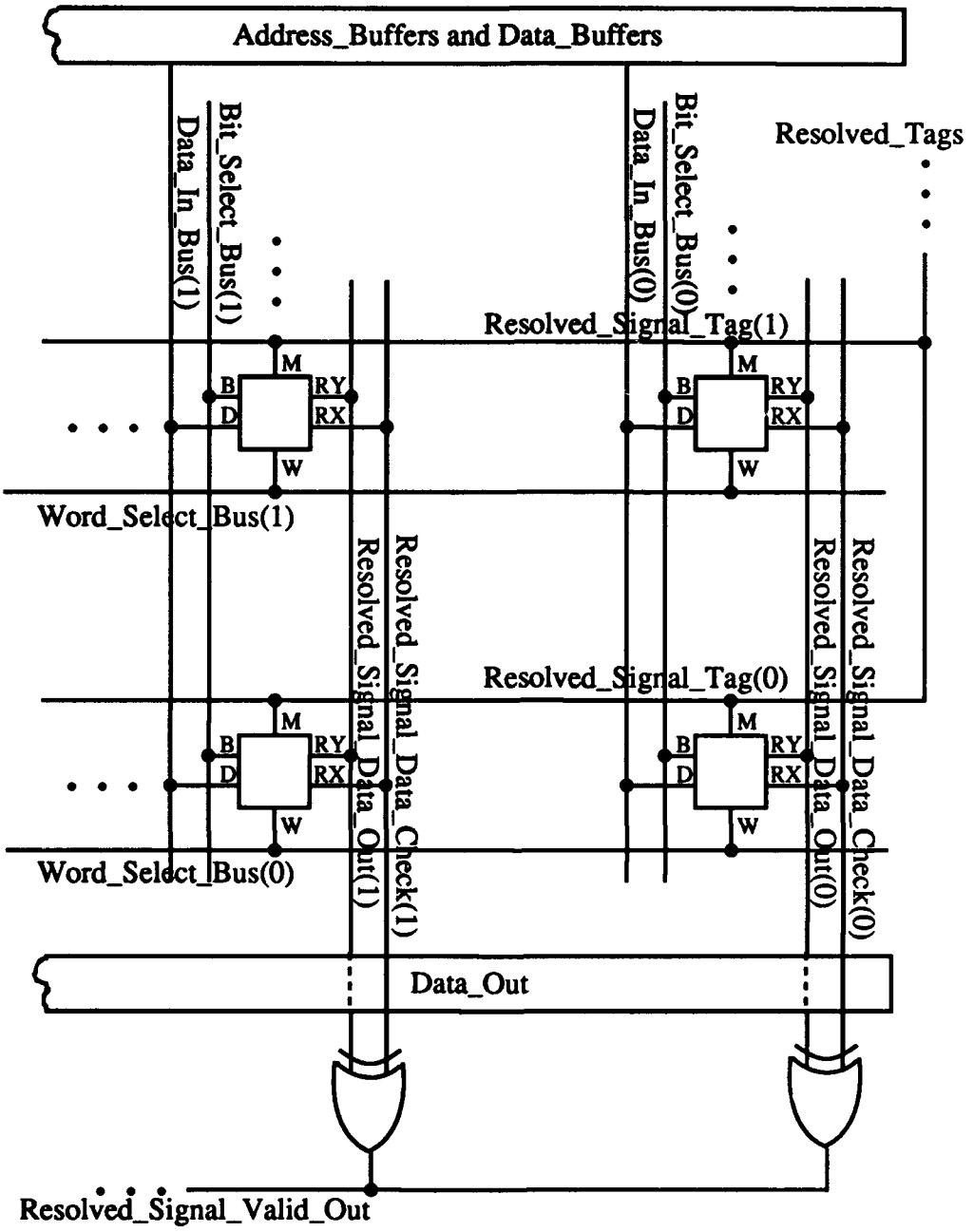


Figure 30. Organization of the CAM Array

the match lines from each cell. If a match is found on all selected bits of a word, then the Resolved_Signal_Tag becomes a '1'. If any of the selected bits do not match the cell

contents, then Resolved_Signal_Tag becomes a '0', meaning the word did not match the data input.

To write to a particular word of the array, a '1' is placed on that word's Word_Select_Bus line. Also, '1's are placed on the Bit_Select_Bus to write to the desired bits of the word.

During a read operation, a single word or multiple words can be read. If a single word is to be read (as will always be the case in the MCC), a '1' is placed on its Word_Select_Bus and all Bit_Select_Bus lines are set to '0'. The word is transported over the Resolved_Signal_Data_Out lines to the Data_Out_Buffers and its complement is transported over the Resolved_Signal_Data_Check line. This line is important during a multiple read because if two or more words are selected to be read, they may not have the same data. The Resolved_Signal_Data_Out performs the wired-OR function for each bit of the word being read. The complement of each bit is resolved on the Resolved_Signal_Data_Check line, which also performs the wired-OR function. The Resolved_Signal_Data_Out and the Resolved_Signal_Data_Check are compared, using exclusive-OR gates, to determine their validity. If the signals are different, then the Resolved_Signal_Data_Out is valid; if the signals are the same, they are invalid. This may more easily be seen in Table 4 below.

Table 4

Truth Table for Validity of Data

RX	RY	P = RX xor RY	
0	0	0	N/A
0	1	1	valid
1	0	1	valid
1	1	0	invalid

For example, suppose a multiple read were performed on the following two 4-bit words (the left-most bit is the most significant bit, bit 3).

1010

0010

The Resolved_Signal_Data_Out of bit 3 would be a '1' (1 + 0 = 1). Resolved_Signal_Data_Check would also be a '1' (1' + 0' = 0 + 1 = 1). Exclusive-ORing the two results produces a '0' (1 XOR 1 = 0), i.e., not valid. On the other hand, Resolved_Signal_Data_Out of bit 2 would be a '0' (0 + 0 = 0) and Resolved_Signal_Data_Check would be a '1' (0' + 0' = 1 + 1 = 1). Exclusive-ORing these two produces a '1' (0 XOR 1 = 1), i.e., valid.

Designing the Controller

Before getting into the details of how the MCC as a whole works, an overview of the MCC controller in an hierarchical fashion is in order. This will allow the reader to learn the terminology used and the figures can then be referenced during the discussion of the operation of the MCC.

Figure 31 shows the highest hierarchical level of the MCC. Appendix D contains the VHDL code describing the interconnects for Figure 31. The cache can logically be viewed as having two parts; 1) the CAM array and its corresponding logic and 2) the controller.

The controller controls the performance of the MCC by communicating with the CAM array. The controller accepts inputs from the array and the MCC's ports and produces outputs to be used by the array. The controller ports are shown in Figure 31. A brief description of each port is presented in Table 5. The ports are in alphabetical order and are identified as either input or output ports.

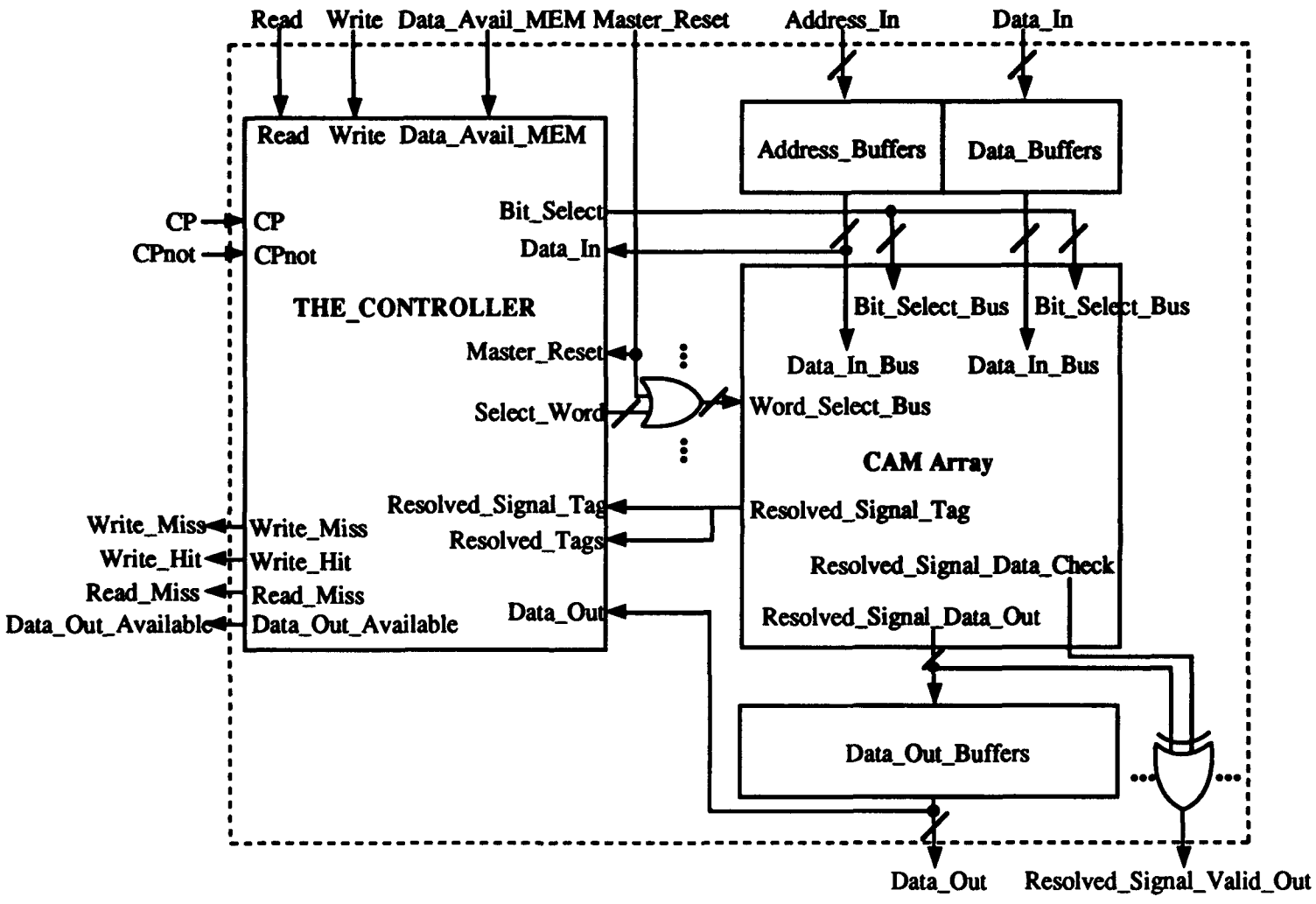


Figure 31. The MCC's Highest Hierarchical Level

Table 5

Description of Controller Ports

Port	Type	Description
Bit_Select	Output	directly connected to Bit_Select_Bus
CP	Input	receives the system clock pulse
CPnot	Input	complement of CP
Data_Avail_MEM	Input	used to signify when the data are available from main memory when a read miss occurs
Data_In	Input	receives the address from the Address_In port
Data_Out	Input	receives the Data _Out vector
Data_Out_Available	Output	signifies that data are present on output port during Read Hit state
Master_Reset	Input	receives MCC's Master_Reset signal that resets the entire chip model
Read	Input	signifies that data are requested of the MCC by the CPU
Read_Miss	Output	signifies that data are not stored on the cache during read function
Resolved_Signal_Tag	Input	receives Resolved_Signal_Tag vector of CAM array signifying which word of CAM array was matched during search operation
Resolved_Tags	Input	receives the Resolved_Tags bit signifying the data searched for are present in the CAM array
Select_Word	Output	outputs data to Word_Select_Bus
Write	Input	signifies when the CPU desires to write data into the MCC
Write_Hit	Output	signifies that data are stored on the MCC during a write function
Write_Miss	Output	signifies that data are not stored on the MCC during a write function

The interconnections between the components that make up the controller are shown in Figure 32. The VHDL code for each of the major components that make up the controller is presented in Appendix C. The purpose of each of these components will now be specified. They are listed in alphabetical order.

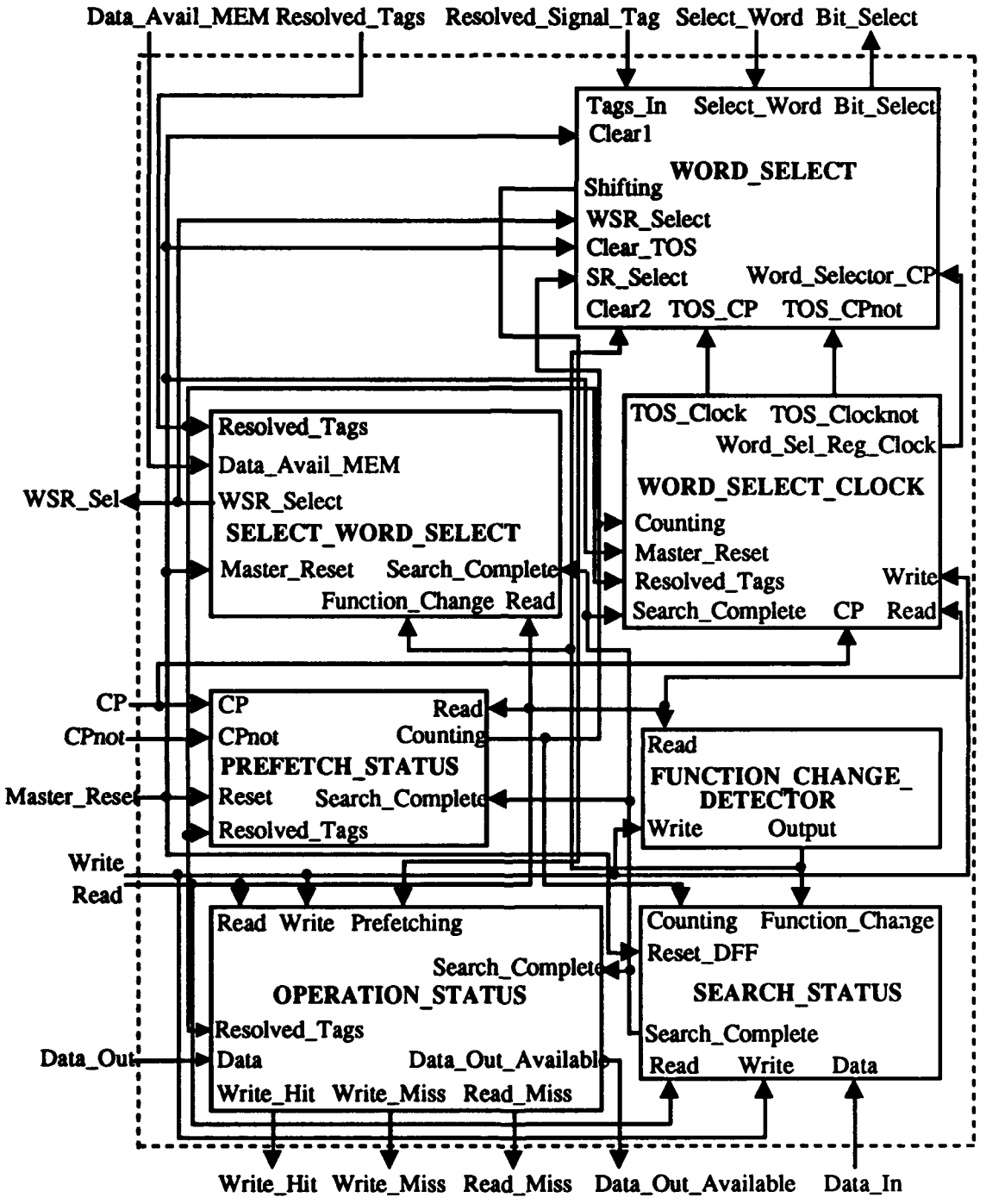


Figure 32. The Controller Components and Interconnections

FUNCTION_CHANGE_DETECTOR: This component determines if the Read or Write ports of the MCC have changed. Figure 33 shows the schematic diagram of the component and Appendix C contains its VHDL code.

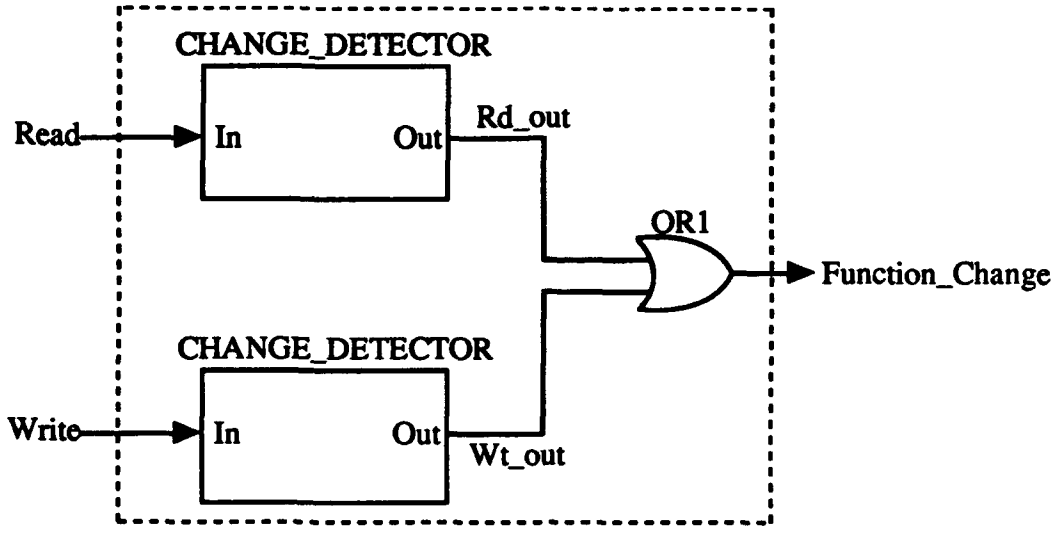


Figure 33. Schematic Diagram of FUNCTION_CHANGE_DETECTOR

The input ports are Read and Write, and the output port is called Function_Change. When Read or Write transitions from '0' to '1' or vice versa, the change detectors detect the change and output a '1' onto the Rd_out or Wt_out signals. These signals are ORed together to produce Function_Change. As a result, if either Read or Write transitions, Function_Change will become a '1'. This is shown in the truth table, Table 6. Function_Change remains a '1' for the time called Change_Detector_Delay. This time is a constant and is defined in chip_pkg.vhd in Appendix E.

OPERATION_STATUS: This component indicates the state the MCC is in. It takes in the incoming signals and from them can determine the state the MCC is in, whether it be the Read Hit, Read Miss, Write Hit, or Write Miss state. The Read Hit state is coded in a less obvious way than the other states. The Data_Out_Available port is asserted high when the data on each read operation in a prefetch cycle is present on the

Table 6

Truth Table for the FUNCTION_CHANGE_DETECTOR Component

Read	Write	Output
0	0	0
0	1	1
1	0	1
1	1	N/A

MCC's Data_Out port when a read hit occurs. The logic diagram of the OPERATION_STATUS is shown in Figure 34. The VHDL code describing this circuit is in Appendix C.

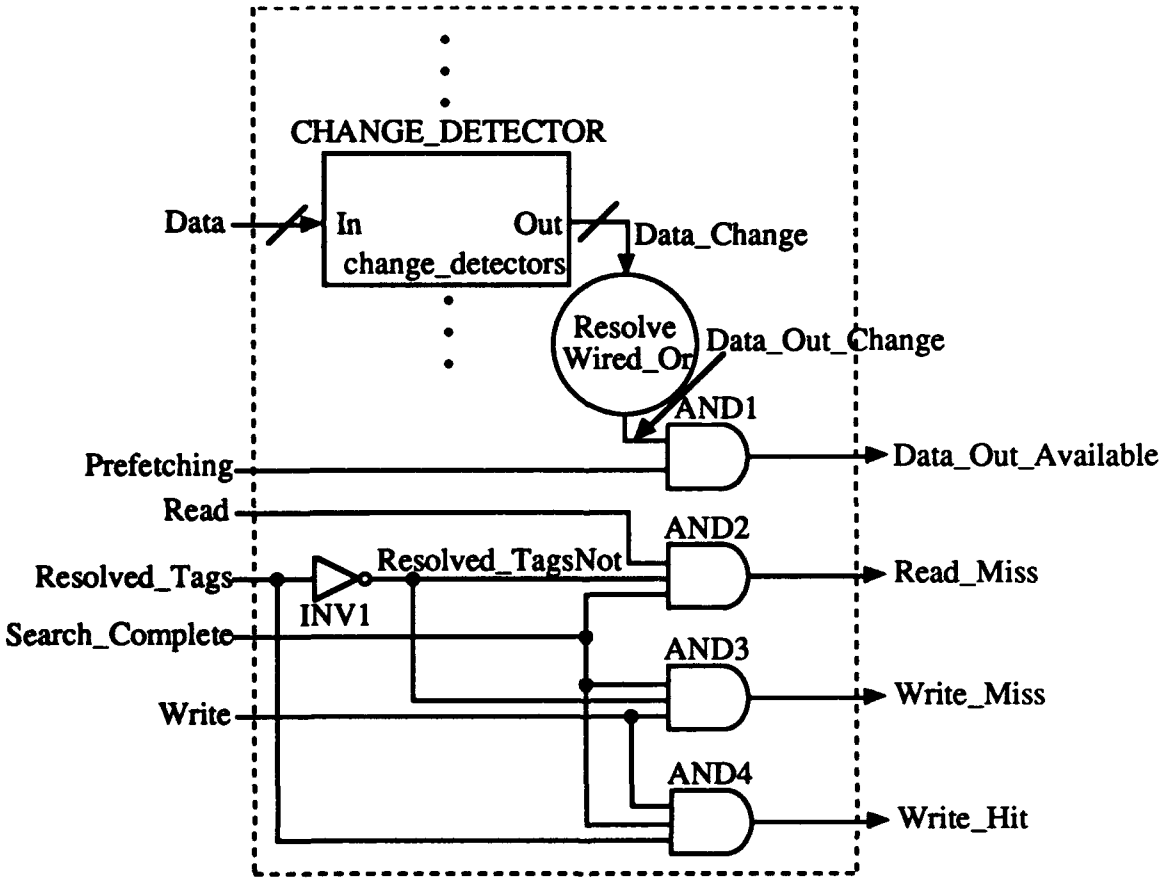


Figure 34. Schematic Diagram of the OPERATION_STATUS Component

The output ports are defined as follows.

• Data_Out_Available: The Data port takes in the Data_Out data of the MCC. Each bit of the Data_Out is connected to a change detector that determines if the bit has changed. The resulting vector is Data_Change. This vector is resolved, using the wired-OR, into one bit, Data_Out_Change. This bit signifies if any of the Data_Out bits have changed. It is ANDed with the signal Prefetching. The AND result is Data_Out_Available, which signifies when the output data of the MCC are available on the MCC's ports. In other words, data are available when new data are available on the MCC's output ports and the MCC is in the prefetching cycle .

• Read_Miss: The Read Miss state is denoted by the AND result of three signals: Read, Resolved_TagsNot, and Search_Complete. Therefore, a read miss occurs when a read is requested, a match was not found (i.e., Resolved_Tags = '0'), and the search for the data is complete.

• Write_Miss: The Write Miss state is denoted by the AND result of the Search_Complete, Resolved_TagsNot, and Write signals. Therefore, a write miss occurs when the search for the data is complete, a match was not found, and a write function was requested.

• Write_Hit: The Write Hit state is denoted by the AND result of the Write, Search_Complete, and Resolved_Tags signals. In other words, a write hit occurs when the write function is requested, the search is complete, and a match was found.

PREFETCH_STATUS: This component produces a '1' on its output port, Counting, only when a read hit has occurred. It remains a '1' for the number of clock cycles required to read the Prefetch_Block_Size (defined in chip_pkg.vhd in Appendix E) from the CAM array. Figure 35 shows the PREFETCH_STATUS component and Appendix C contains its VHDL code.

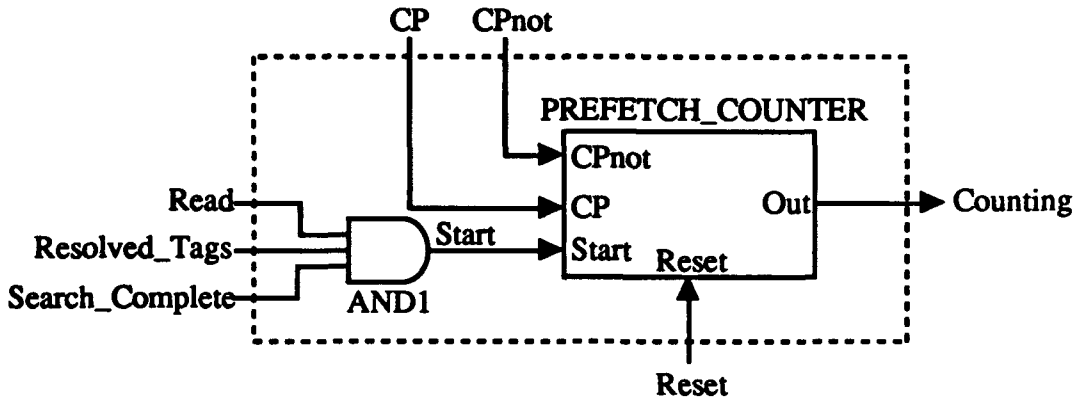


Figure 35. Schematic Diagram of the PREFETCH_STATUS Component

The component PREFETCH_COUNTER counts to the number of clock cycles equal to the Prefetch_Block_Size. While counting, it produces a '1', which is put onto Counting port. It starts this counting cycle after a search is complete, the data are found, and a read is requested of the cache. The systems clock (CP) then clocks the counter inside PREFETCH_COUNTER. When Prefetch_Block_Size is reached, Counting becomes a '0' and the prefetch cycle is complete.

SEARCH_STATUS: This component provides enough delay to accomplish a search and produces a '1' on its output port, Search_Complete, until it is reset to '0'. The schematic diagram is shown in Figure 36 and its VHDL code is in Appendix C.

This is how SEARCH_STATUS works. The input into the D-type flip-flop is the signal Search_Done. Search_Done is defined as:

Search_Done = (Resolved_Signal_Address_Change)(Read + Write).

Thus, when the address portion of the Data_In_Bus changes and a Read or a Write is requested of the MCC, a '1' is clocked into the D-type flip-flop. Notice that Search_Done clocks itself into the D-type flip-flop.

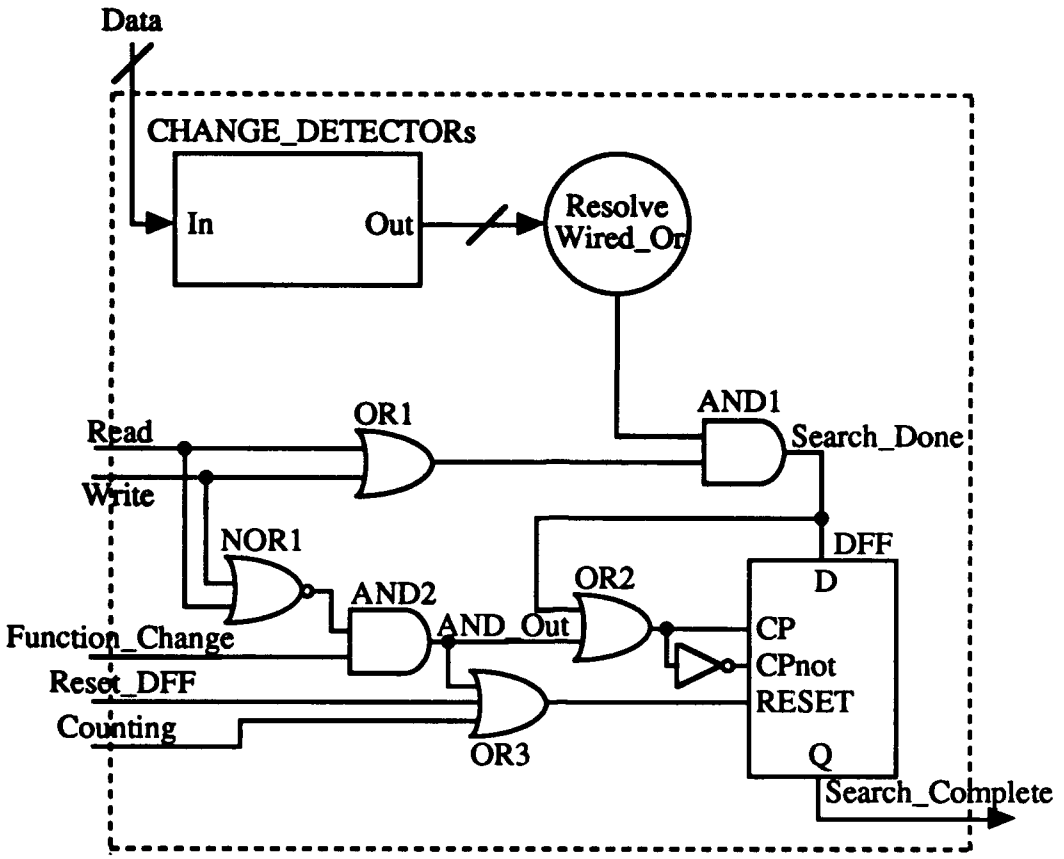


Figure 36. Schematic Diagram of the SEARCH_STATUS Component

The D-type flip-flop is reset with the OR result of three signals: AND_out, Reset_DFF, and Counting. AND_out is defined as:

$$AND\_out = (Function\_Change)(Read + Write)'.$$

This causes the D-type flip-flop to be reset only when the Read or Write signal transitions from a '1' to a '0'. Thus, the D-type flip-flop is reset at the end of a Read or Write request.

The second way the D-type flip-flop can be reset is with the Reset_DFF port. This port is connected to the MCC's Master_Reset port. Therefore, when Master_Reset is high, the D-type flip-flop is reset to '0'.

The third way to reset the D-type flip-flop is with the Counting signal. When Counting is '1', the MCC is in the prefetch cycle. If the prefetch cycle ends and Search_Complete is still a '1', the prefetch cycle will start again. To avoid this situation, the D-type flip-flop is reset to '0' upon activation of the prefetch cycle.

SELECT_WORD_SELECT: This component is used to select the register of the WORD_SELECT component that will be connected to the Word_Select_Bus. It outputs a '0' or '1', which is ported to the multiplexers that select the data to be put onto the Word_Select_Bus (see WORD_SELECT below). The schematic diagram is shown in Figure 37 and the VHDL code is in Appendix C.

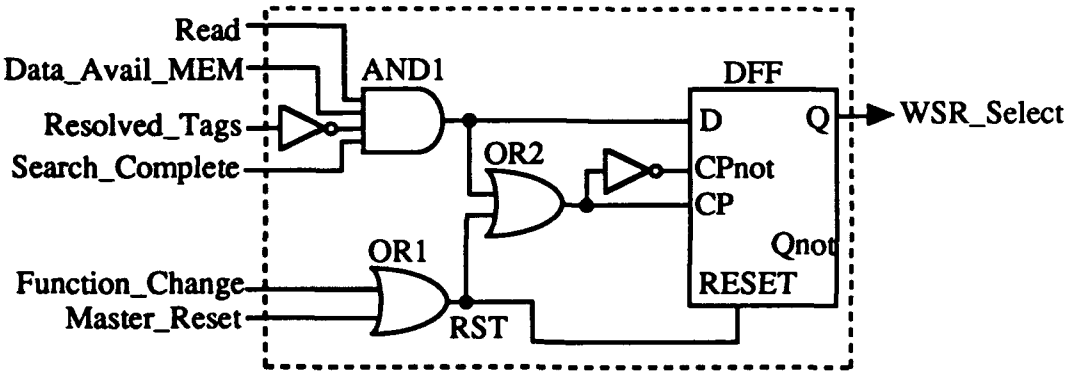


Figure 37. Schematic Diagram of the SELECT_WORD_SELECT Component

When the output of this component, WSR_Select, is '0', the SHIFT_REGISTER outputs of WORD_SELECT are connected through the multiplexers to the Word_Select_Bus. SHIFT_REGISTER shifts upward on the prefetch cycle during a read hit to prefetch the desired block of data. It is also used to write to a particular word of the CAM array on a write hit.

When WSR_Select is a '1', the TOS_SHIFTER's output is connected through the multiplexers of WORD_SELECT to the Word_Select_Bus. This register is used to keep track of where the top of the stack is so data can be written into the MCC in the order

they are used by the CPU. Therefore, on a read miss, the data are written into the CAM array using TOS_Shifter output.

Thus, we have seen that the SELECT_WORD_SELECT selects which register's outputs will go onto the Word_Select_Bus. The SELECT_WORD_SELECT's output, WSR_Select, is connected to the multiplexers of the WORD_SELECT component. A D-type flip-flop stores the value used as the WSR_Select. The only time WSR_Select is a '1' is when a read miss has occurred and data from main memory will be written into the CAM array. This occurs when Read, the complement of Resolved_Tags, and Search_Complete are high. The one thing still missing is a signal from the main memory telling the MCC that data are available on the data buses. This signal is Data_Avail_MEM. When this signal is asserted high, the D input of the D-type flip-flop gets a '1' from the output of the AND1 gate, which is also used as the clock input. Therefore, when a read is requested, a read miss occurs, the search is complete, and the data are available from main memory, the WSR_Select output becomes a '1'.

WSR_Select is a '0' at all other times. As the MCC is initialized, the Master_Reset port is asserted high. This signal is connected to the RESET port of the D-type flip-flop and is therefore reset to '0'. Also, the Function_Change port is connected to the output of FUNCTION_CHANGE_DETECTOR. Therefore, when this port is asserted high, the D-type flip-flop is reset to '0'.

WORD_SELECT: This component is the most complex of the controller components. Its purpose is 1) to output data onto the Word_Select_Bus and 2) provide data to the Bit_Select_Bus. The schematic diagram is shown in Figure 38 and its VHDL code is located in Appendix C.

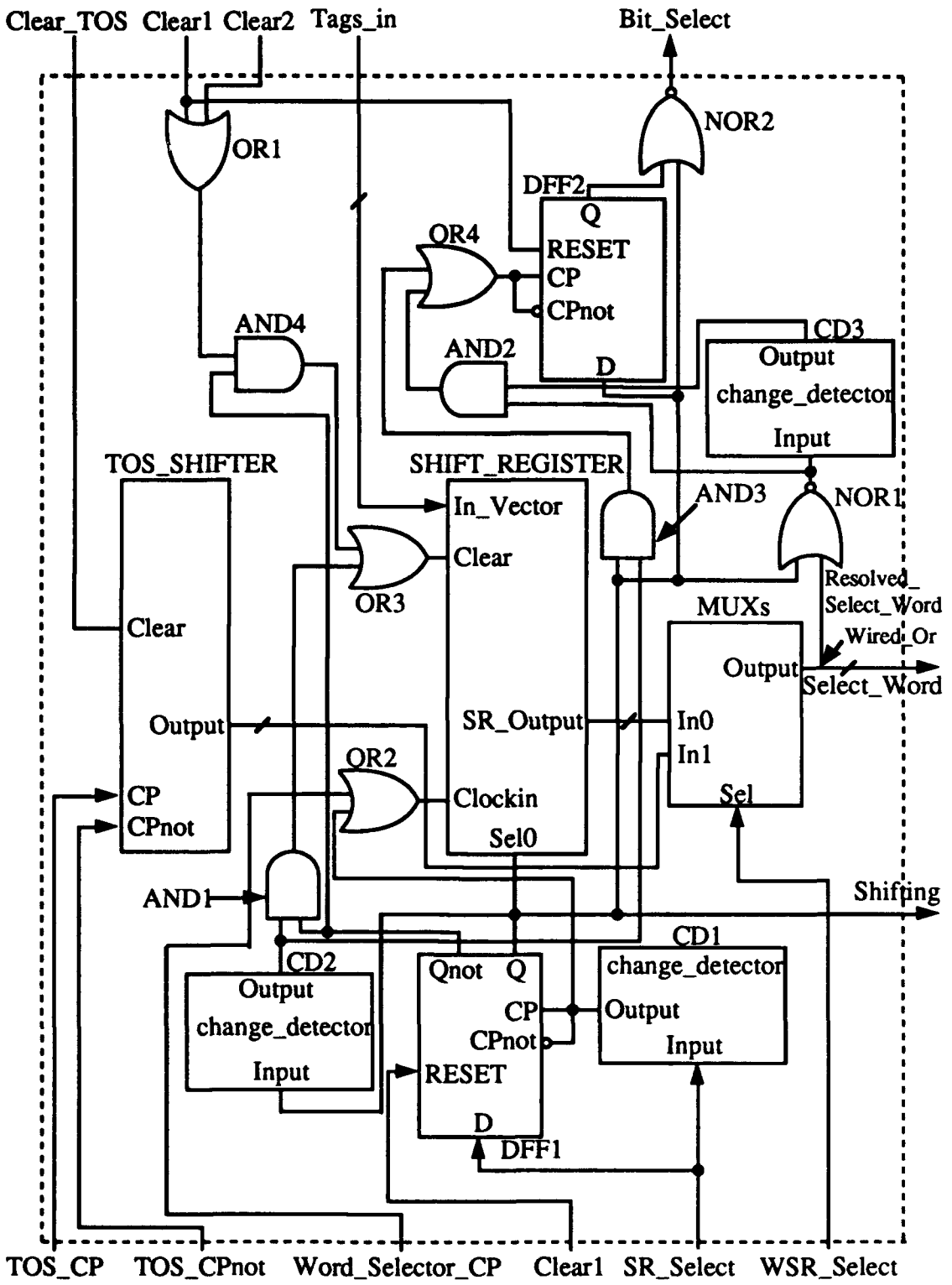


Figure 38. Schematic Diagram of WORD_SELECT Component

One thing to notice about the operation of this MCC is that in order to write to the TOS in the Read Miss state, the TOS position must be held from the previous write operation. Also, in the Read Hit state, a pointer is shifted upward to prefetch the data in the order in which they were written in order to capture structural locality. These two states are in direct conflict for the CAM pointer. The problem is solved by placing not one shift register on the MCC but two. One is used to keep track of the TOS position for writing into the cache during the Read Miss state. The other is used to locate the data during a Read Hit and to shift upward to prefetch the contents of the CAM array. 2x1 multiplexers are used to select which shift register's output is used as the word select input for the CAM array.

The TOS_SHIFTER is used to store the top-of-stack position for the write operation during the Read Miss state. The SHIFT_REGISTER is used to update data during the Write Hit state and also to shift upward during the prefetch cycle in the Read Hit state. The outputs of these two shift registers are connected to a series of multiplexers, with the In0 ports of each multiplexer connected to the bits of the SHIFT_REGISTER output and the In1 ports connected to the bits of the TOS_SHIFTER output. The Sel ports of the multiplexers are connected to the WSR_Select port, which determines the shift register that will be connected to the Word_Select_Bus. WSR_Select is connected to the output of the SELECT_WORD_SELECT component.

The operation of WORD_SELECT is fairly straightforward when TOS_SHIFTER is selected. This register is only selected during the Read Miss state. It is during this state that the CAM array does not contain the data asked for by the CPU. When this occurs, main memory will provide the CPU with the required data and the MCC will intercept these data and write them into the CAM array. The Clear_TOS port of the TOS_SHIFTER clears the register and initializes one of the flip-flops to '1' to act as the

first entry of the top-of-stack. It is cleared and initialized only with the MCC's Master_Reset. After initialization, the register is shifted on the clock pulse. The inputs into CP and CPnot come from the component WORD_SELECT_CLOCK described later.

The operation of WORD_SELECT is much more complicated in the Write Hit and Read Hit states when the SHIFT_REGISTER is selected. This register has a port called Sel0 that determines whether it will load incoming data or shift upward. During both Write Hit and Read Hit states the register will load data. During the Read Hit state, the register first loads the data then enters the prefetch cycle when it is shifted upward to read the desired prefetch block size. It is critical at the end of the prefetch cycle that the Bit_Select port not change to '1' until after SHIFT_REGISTER is cleared and Select_Word becomes all zeros. If it does change to '1', an inadvertent write will occur. To avoid this, the Select_Word port's data are resolved, using wired-OR, into a single bit. When this bit changes to a '0' (i.e., Word_Select_Bus is all zeros) and SR_Select is stored as a '0' (i.e., Counting is a '0' and no prefetching is occurring) in D-type flip-flop DFF1, SR_Select (currently '0') is clocked into DFF2. The output of DFF2 is NORed with the output of DFF1 (SR_Select) and the result is ported to the Bit_Select_Bus. The result in this case is '1' since '0' NOR '0' is '1'. Therefore, since Select_Word is connected to the Word_Select_Bus (currently all zeros), the write operation is disabled.

SHIFT_REGISTER is cleared only when the prefetch cycle is not occurring, i.e., when SR_Select = '0' is stored in DFF1. The Clear2 port is connected to the signal Function_Change. So, as long as Counting is '0' (i.e., the Qnot output of DFF1 is '1'), the AND4 output is a '1' after either the Read or Write ports change. The output of AND4 is connected through OR3 into the Clear port of SHIFT_REGISTER, thus clearing the register. SHIFT_REGISTER is also cleared promptly after the prefetch cycle. When SR_Select changes to a '0', it is clocked into DFF1. The output of DFF1 will then

change causing the CD2 output to go to '1' for a period of time. Qnot of DFF1 is ANDed with the output of CD2, and this signal is connected through OR3 to the clear port of SHIFT_REGISTER.

We saw above how Bit_Select becomes a '1' after the Read Hit state is completed. Now let's discuss how it becomes a '0' during that state. When the output of PREFETCH_STATUS, Counting, becomes a '1', we want Bit_Select to become a '0' to allow for the read operation. As discussed above, Counting is connected to the SR_Select port of WORD_SELECT. When Counting changes to '1', it is clocked into DFF1 and the output of DFF1 is connected to NOR2. Thus, NOR2 produces a '0' and is output onto the Bit_Select port, which in turn forces the Bit_Select_Bus to become '0's.

Let's summarize this complicated circuit. The reader may refer to the VHDL code of THE_CONTROLLER in Appendix D for the port connections of WORD_SELECT. The circuit is initialized when Master_Reset is asserted high. The Clear port of TOS_SHIFTER is connected to Master_Reset through Clear_TOS port. Therefore, when Master_Reset goes high, TOS_SHIFTER is cleared and one flip-flop is set to '1' to act as the TOS pointer. At the same time, SHIFT_REGISTER is cleared to all zeros. Clearing this register is a bit more complex than initializing TOS_SHIFTER. The first thing that must be done is to reset DFF1. This is accomplished with Master_Reset. After resetting, Qnot of DFF1 is '1' and is ANDed with OR1 output. One of the inputs to OR1 is Master_Reset so the AND result is '1'. This signal is then connected to OR3 and the OR3 output is connected to the Clear port of SHIFT_REGISTER. Thus, SHIFT_REGISTER is cleared. The other way SHIFT_REGISTER is cleared is when the signal, Function_Change, is asserted high when the prefetch cycle is not occurring. The signal, Function_Change, is connected to the Clear2 port of WORD_SELECT. If it is '1', then the output of OR1 is '1'. This signal is ANDed with the Qnot output of

DFF1. Qnot is a '1' as long as the MCC is not in the prefetch cycle. Therefore, the AND result is a '1', which goes through OR3 to clear SHIFT_REGISTER.

If a read miss occurs, the MCC will write the data into the CAM array when it is available from main memory. In this case, the WSR_Select port becomes a '1' selecting the TOS_SHIFTER and shifting it upward using the CP and CPnot inputs. The Bit_Select port is already a '1' from initialization or from the previous state change. Thus, the CAM array is set up for a write and the write is accomplished.

If a write hit occurs, the Resolved_Signal_Tag showing the word that was found is loaded into SHIFT_REGISTER. WSR_Select is a '0', which selects the SHIFT_REGISTER data and outputs the data onto the Word_Select_Bus. Thus, the CAM array is again setup for a write operation.

If a read hit occurs, SR_Select becomes a '1' and is stored into DFF1. The Q output of DFF1 then selects the shift operation of the SHIFT_REGISTER and on each clock pulse it is shifted upward until the prefetch block size is reached. Q is also input into NOR2 whose output becomes a '0', which goes onto the Bit_Select_Bus. Thus, data are read in a FIFO manner from the CAM array on each clock pulse.

WORD_SELECT_CLOCK. The purpose of this component is to act as the clock inputs for the registers in the WORD_SELECT component. The schematic diagram is shown in Figure 39 and its VHDL code is in Appendix C.

This component has three output ports. One of these outputs, Word_Sel_Reg_Clock, is connected to the SHIFT_REGISTER of the WORD_SELECT component. It clocks this register during the prefetch cycle to load incoming data into the register. The other two output ports, TOS_Clock and TOS_ClockNot, are connected to the TOS_SHIFTER of WORD_SELECT. These two signals are the clock inputs of

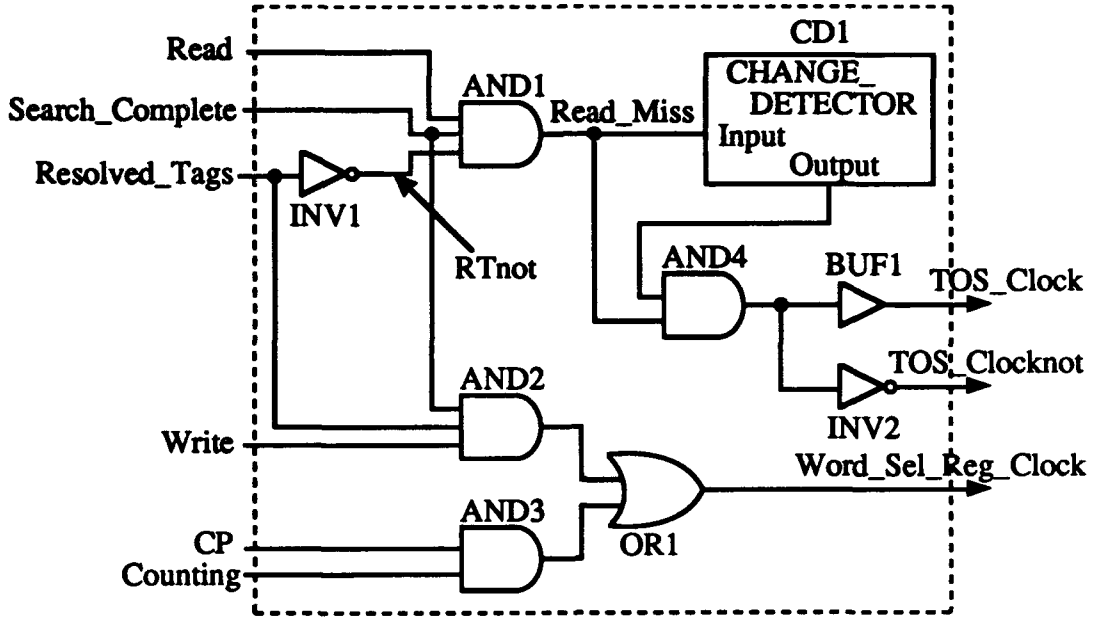


Figure 39. Schematic Diagram of the WORD_SELECT_CLOCK Component

TOS_SHIFTER to shift the register prior to the write operation during the Read Miss state.

The output Word_Sel_Reg_Clock is determined by the following boolean equation:

Word_Sel_Reg_Clock = (Search_Complete)(Resolved_Tags)(Write) + (CP)(Counting).

This means that when a search is complete, a match was found in the CAM array, and a write function was requested, or when the systems clock pulses during a prefetch cycle (i.e., Counting = '1'), Word_Sel_Reg_Clock is asserted.

The output TOS_Clock needs to have a rising and falling edge to act as the clock input for the TOS_SHIFTER. Therefore, a change detector was placed on the component. When the signal Read_Miss changes from '0' to '1' or vice versa, the change detector outputs a '1' for a short period of time. Read_Miss is determined as follows:

Read_Miss = (Read)(Search_Complete)(RTnot).

Thus, when a read is requested, the search is complete, and the search was unsuccessful, Read_Miss becomes a '1'. This signal is ANDed with the output of the change detector to pulse TOS_Clock. Therefore, TOS_Clock pulses only when Read_Miss transitions to '1'. TOS_ClockNot is the complement of TOS_Clock. The buffer, BUF1, is needed to ensure TOS_Clock and TOS_ClockNot transition at the same time.

Putting it All Together

With the four states of the MCC in mind, the logic and control lines were drawn schematically to represent the operations the MCC was to perform. Therefore, a structural gate level design was integrated onto the chip model to control the CAM array. Table 7 shows the necessary inputs into the CAM array to accomplish each operation for the various states. All of the components that make up the MCC were discussed in some detail above. Refer to those sections for further clarification of the signal and component names.

Table 7

CAM Array Inputs for the Main CAM Cache States

	Bit_Select_Bus	Word_Select_Bus
Read		
Search	1	0
Hit		
Read	0	Resolved_Signal_Tag
Miss		
Write	1	TOS_Output
Write		
Search	1	0
Hit		
Write	1	Resolved_Signal_Tag
Miss	N/A	N/A

As stated before, each function begins with the search operation. This operation requires that the Bit_Select_Bus be asserted high and the Word_Select_Bus be asserted low. If, during a read, the search is successful, the MCC goes into the Read Hit state and the read operation is performed. A read requires the Bit_Select_Bus to be asserted low and the word being read must have a '1' asserted on its Word_Select_Bus lines. Alternatively, if the search is unsuccessful, the Read Miss state is entered. This state requires a write operation in which the Bit_Select_Bus must be asserted high and the Word_Select_Bus gets the data from the TOS_SHIFTER of the WORD_SELECT component. The TOS_SHIFTER holds the next stack position for a write.

If the search is successful during a write function, then the Write Hit state is entered and a write operation is performed. During a write operation, the Bit_Select_Bus must be asserted high. The Word_Select_Bus receives the Resolved_Signal_Tag data indicating the word that was found. If the search were unsuccessful, the Write Miss state is entered and no further action is required.

The operation of the entire MCC will now be discussed. Since the Read and Write functions require a search, the search will first be explained. The remainder of this section describes the functionality of the MCC in each of the four states.

The Search. According to Table 7, the Bit_Select_Bus must be asserted high and the Word_Select_Bus must be asserted low to accomplish the search. The Bit_Select_Bus gets its data directly from the Bit_Select port of the controller. This port is '1' as long as the MCC is not prefetching in the Read Hit state. The Word_Select_Bus gets its data indirectly from the SHIFT_REGISTER (see Figure 38). This register uses the D-type flip-flop with Reset. As long as the MCC is not in the prefetch mode, a change either in the Read or Write port will cause the SHIFT_REGISTER to reset. When the Read or Write ports change, the FUNCTION_CHANGE_DETECTOR detects the

transition and clears the register to all zeros. Note that after each function is complete (either the read or write function), the SHIFT_REGISTER is reset. Thus, when new data are to be operated on during a read or write function, Word_Selector will have been reset to zeros and will be ready for another search operation. Also note that SHIFT_REGISTER is reset upon the initialization of the MCC by the Master_Reset signal.

As mentioned above, the Word_Select_Bus gets its data indirectly from SHIFT_REGISTER during a search operation; 2 x 1 multiplexers are located between the two (see Figure 38). The output data of SHIFT_REGISTER are fed into the In0 ports of the 2x1 multiplexers. Consequently, the multiplexers must have a '0' value at the Sel ports before the data are output onto the Word_Select_Bus. This is done by the SELECT_WORD_SELECT component shown in Figure 37 which monitors the change in the Read port of the MCC. When the Read signal transitions from '1' to '0', FUNCTION_CHANGE_DETECTOR outputs a '1' onto signal Function_Change, which is connected to the Reset port of an edge-triggered D-type flip-flop in SELECT_WORD_SELECT. This resets the flip-flop to '0'. Its output, WSR_Select, is connected to the Sel ports of the multiplexers in WORD_SELECT, thereby selecting the data from SHIFT_REGISTER. Upon initialization, the Master_Reset signal also resets this D-type flip-flop. Consequently, the MCC is set up for a search operation after initialization.

The Read Hit. To do a read, the Read port of the MCC must indicate that a read function is requested and the address to read must be available. If the search operation, as explained above, is successful, then the MCC enters the Read Hit state and begins its prefetching cycle. Only one bit of the Resolved_Signal_Tag vector will be '1' on a search hit. This is because an address will never appear more than once in the CAM ar-

ray since during a write function, a search is performed first, and if successful, the new data are written over the old.

The Resolved_Signal_Tag lines are connected to the In_Vector port of SHIFT_REGISTER through the Tags_In port of WORD_SELECT (see Figures 32 and 38). These data are stored into the SHIFT_REGISTER by clocking them in with the CP (the MCC's clock port) ANDed with the Counting signal (see the WORD_SELECT_CLOCK component in Figure 39). They are ANDed to ensure the SHIFT_REGISTER is clocked only when the MCC is prefetching. The SHIFT_REGISTER's outputs are selected by the multiplexers and the data are output into the CAM array.

After SHIFT_REGISTER is loaded, the Sel0 port of SHIFT_REGISTER is changed to a '1'. This is accomplished with the Counting signal. When it changes to a '1', it clocks itself into DFF1, shown in Figure 38. This signal is connected to the Sel0 port of SHIFT_REGISTER. A '1' on this port switches its functionality from loading to shifting. Since its Clockin port is connected to the CP port of the MCC and ANDed with the Counting signal, the register will shift upward the number of times defined in the prefetch block size, allowing data from the CAM array to be read from the MCC in the order in which they were written.

Another requirement for the read operation is that the Bit_Select_Bus be all '0's. This is done by using the output of DFF1 of WORD_SELECT (Figure 38). When it changes to a '1', it clocks itself into DFF2. The output of DFF2 is then NORed with the output of DFF1, which is essentially the Counting signal with a delay. The Counting signal comes from the PREFETCH_COUNTER that is stimulated by the Resolved_Tags, Read, and Search_Complete (a timing signal signifying the completion of a search) signals ANDed together. The result of the AND signifies a read hit has occurred. The

PREFETCH_COUNTER produces a '1' and asserts the signal Counting high as long as the binary counter inside the PREFETCH_COUNTER does not equal the predefined prefetch block size. The binary counter counts up by one on each clock pulse. The counter's output is compared with the prefetch block size. When they are equal, the PREFETCH_COUNTER produces a '0' on the Counting signal, meaning the prefetch cycle is complete.

While Counting is '1', the MCC's clock is able to clock the SHIFT_REGISTER. On each clock pulse, the register is shifted up by one and the contents of that CAM word are read. Thus, the predefined number of words is prefetched, each word being read on the clock pulse.

The Read Miss. If a miss occurs on a read, then the address and its corresponding data need to be written into the CAM. The CAM is written to in a FIFO manner. Therefore, the TOS_SHIFTER will be used to select the word to be written. The only purpose of this shift register is to keep track of the top of the stack. In the MCC, this register is shifted just before the write to the CAM. This is done by merely clocking the register. The clock input is the AND result of the Read, Search_Complete, and Resolved_Tags' signals (see Figure 39). Those signals ANDed together signify a read miss. Since a complete clock cycle ('0' to '1', then '1' to '0') is needed to shift the TOS_SHIFTER, a CHANGE_DETECTOR was used to detect the transition of the AND result. The CHANGE_DETECTOR produces a '1' for a short period of time then returns to '0', therefore completing its own clock cycle.

The Sel port of the multiplexers in the WORD_SELECT component (Figure 38) is determined by the SELECT_WORD_SELECT component. The AND result of the Read, Search_Complete, Resolved_Tags', and Data_Avail_MEM signals is connected to the input of the D-type flip-flop of the SELECT_WORD_SELECT and (after going

through an OR gate) is also connected to the flip-flop's clock port. Thus, the AND result clocks itself into the D-type flip-flop. The output of the flip-flop is connected to the Sel port of the multiplexers, thus selecting the TOS_SHIFTER. The Bit_Select_Bus is already set to '1's, so the write occurs as soon as the data are present on the Address_In and Data_In ports.

After this read function is complete, the Read signal is asserted low and the FUNCTION_CHANGE_DETECTOR detects this change and outputs a '1'. This '1' then goes through the Clear2 port of WORD_SELECT and resets the SHIFT_REGISTER. This sets the MCC up for the next search operation. Now, let's take a look at the write function.

The Write Hit. The write function, as mentioned before, also starts with the search operation. In this case, the Write signal is asserted high and the address and data are available on the Address_In and Data_In ports, respectively.

The search is performed and, if successful, the Write Hit state is entered. In this state, the word that matched the input address will have its data replaced by the new data. The word is selected by the Resolved_Signal_Tag vector (this vector signifies which word is matched) being loaded into the SHIFT_REGISTER of WORD_SELECT. The Resolved_Signal_Tag vector is connected to the In_Vector port of this register (see Figure 38). The AND result of the Write, Search_Complete, and Resolved_Tags signals is used to clock the inputs into the SHIFT_REGISTER (see the WORD_SELECT_CLOCK in Figure 39). This AND result signifies a search hit on a write function. The output of SHIFT_REGISTER is input into the CAM array on the Word_Select_Bus through the multiplexers. Since the Bit_Select_Bus is already set up to do a write (i.e., the Bit_Select_Bus lines are all '1's), the write operation is performed on the selected word and the old data are replaced with the new.

The Write Miss. If, on a write function, the search is unsuccessful, the Write Miss state is entered. In this state, the MCC simply asserts the Write_Miss port high for a short period of time and then waits until called upon again.

Summary

This chapter detailed the design of the MCC from its functionality through the final integration of the CAM array and the controller. The VHDL model of the chip was designed and built from the bottom up. The CAM cell design was taken from DeCegama (3:87) and implemented in VHDL. The CAM cell was then used to build the CAM array, and the logic associated with the CAM array was integrated onto the chip model. Basic components such as flip-flops and shift registers were needed in order to put the rest of the model together. Once these basic components were built and tested, larger components were assembled. These larger components were pieced together to form the controller and the controller was integrated onto the chip model. A VHDL description of the main CAM cache used to exploit structural locality was the final product.

IV. Testing and Analysis

Overview

In Chapter 3, the inner workings of the MCC were discussed in detail. Let's now take an overview at the MCC and analyze its behavior. This chapter contains a look at the behavior of the MCC and the testing of the chip model to verify its functionality. The context of how the MCC fits into a memory subsystem will be discussed. The performance of the MCC as to the timing of the operations in each of the four states is described. A detailed analysis is then presented including critical-path calculations as well as a critical view on how to improve the MCC's performance and space utilization. Finally, some issues that need to be considered during the fabrication of the chip conclude the chapter.

Behavior of the MCC

As described in Chapter 3, the MCC takes on one of four distinct states: the Read Hit, Read Miss, Write Hit, and Write Miss states. Depending on the function requested (read or write) and the data available, the MCC will enter one of these states. This is shown in the VHDL behavioral description of the MCC located in Appendix D. Figure 40 shows a diagram of the states and how the states are entered.

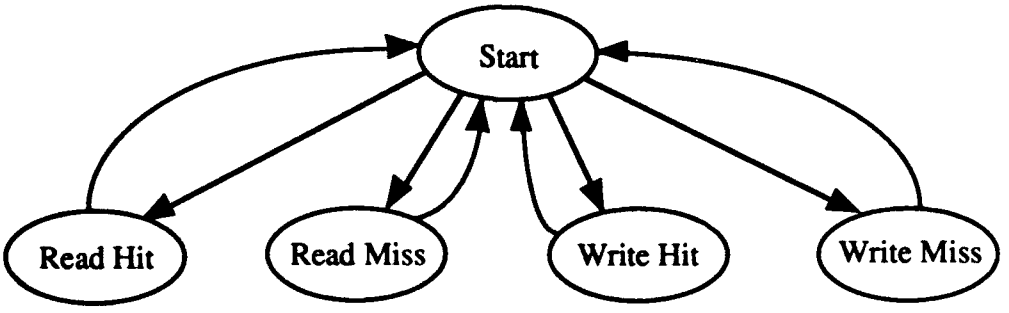


Figure 40. Sate Diagram of Main CAM Cache

Figure 41 shows the flowchart of how the chip model works. Regardless of whether a read or a write function is requested of the MCC, a search is performed first. After the search is performed, one of the four states shown in Figure 40 will be entered.

When the CPU requests a read and provides the address, either the Read Hit or Read Miss state will be entered. If the address is found in the CAM array, the Read Hit state is entered. This is the only state that works synchronously. When a read hit occurs, the MCC begins to prefetch memory locations in the order in which they were written into the cache. On each clock pulse, the SHIFT_REGISTER is shifted upward resulting in the next array position in the CAM to be read. As each new address and associated data become available on the MCC's output ports, a signal called Data_Out_Available is asserted high, signifying that the data are available.

If the address is not present in the CAM array, the MCC asserts high the Read_Miss port signifying that a read miss has occurred. The MCC then goes into a wait state until main memory has placed the requested data on the data bus. The MCC must be notified, through the Data_Avail_MEM port, that the data are available on the bus from main memory. The data are then written onto the TOS of the CAM array.

When the CPU requests a write, the address and data are also provided. If the address is present in the CAM array, the Write Hit state is entered. During this state, the Write_Hit port is asserted high and its current data are replaced with the new data. If the address in not stored in the MCC, the Write Miss state is entered and the MCC simply asserts its Write_Miss port signifying that a write miss occurred and waits for the next function.

Testing

Each component of the VHDL model was thoroughly tested to ensure that its functionality corresponded to the expected behavior. Throughout the initial testing of

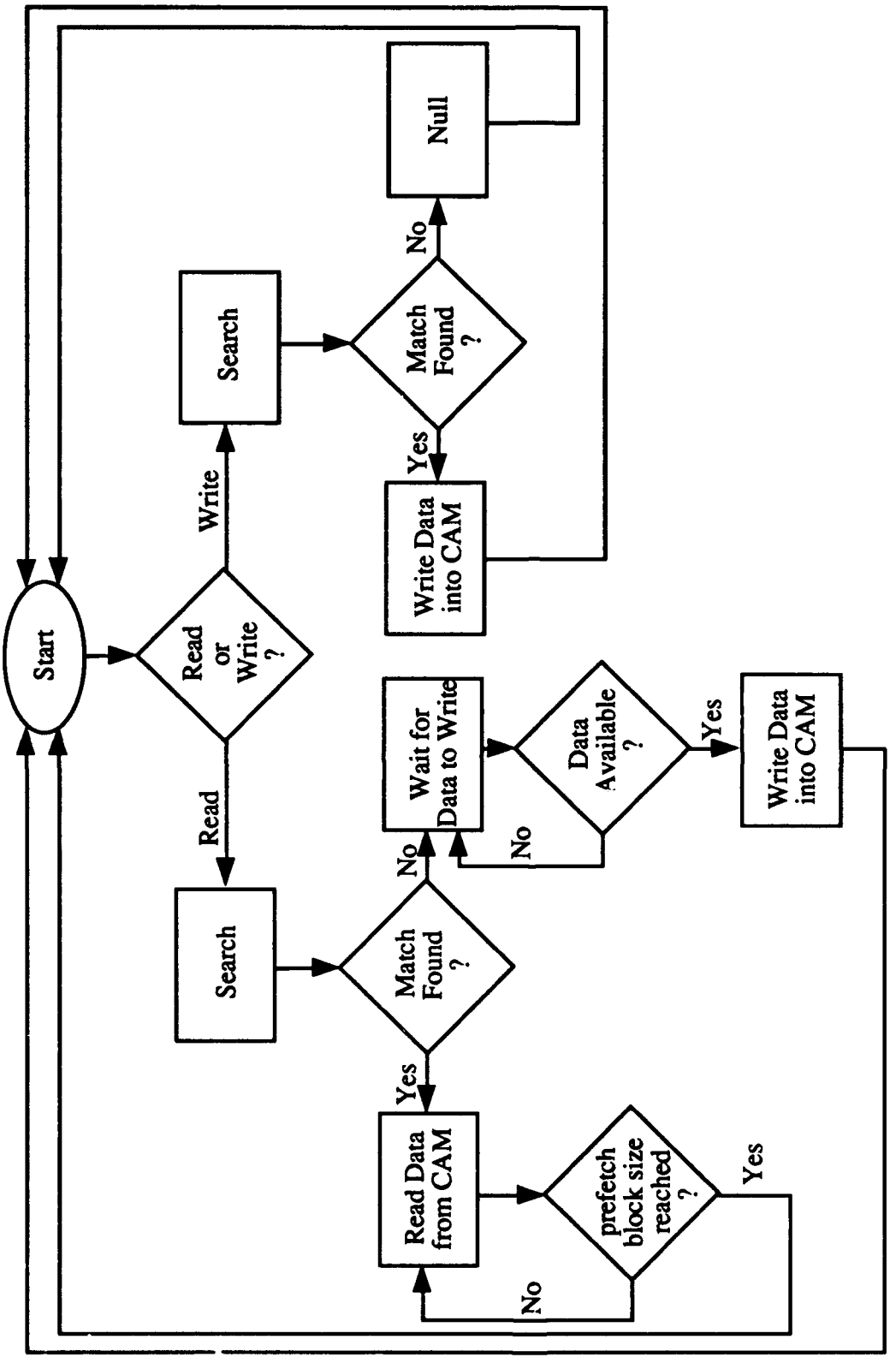


Figure 41. Flowchart of MCC

each component and the MCC itself, the word length was set at 6 bits, composed of 3 bits for the address and 3 bits for the data. The CAM depth was also initialized at 3. This was sufficiently large enough to ensure the model was working correctly, but small enough to allow the simulations to run very fast.

Appendix F contains the code used to exercise the MCC model as well as a simulation run including inputs and outputs. The appendix has the file used to stimulate the MCC, the test bench, and the configuration file. The simulation run contains a simulation of each of the four states. This data is part of the data used to analyze the performance and find the critical paths through the MCC.

A small memory system is shown in Figure 42 and the VHDL code is in Appendix G. This system consists of the MCC connected to a CPU and main memory (MEM) by buses and control lines. The only purpose of the CPU model is to produce addresses and request the memories to read or write the address. The sole purpose of MEM is to produce data corresponding to the address from the CPU and to produce data during the Read Miss state of the MCC. This system was built around the MCC to test the MCC only. It is not how an actual memory system works. The word length used in these tests was 64 bits; 32 for the address portion and 32 for the data portion. The CAM depth was set at 10 words during one test and 15 during another.

Figure 42 shows how the simple memory system is configured. The CPU generates an address and a memory request every 100 ns. If the request is a write, MEM automatically produces data onto the data bus (Data_Sig in the figure and in the code). This makes it appear to the MCC that the CPU produced both the address and the data.

Let's look at how the system works. When the CPU issues a read request, it also places an address on the address bus (Address_Sig). The MCC then searches for the address and if a hit occurs, the Read Hit state is entered and the prefetch cycle is

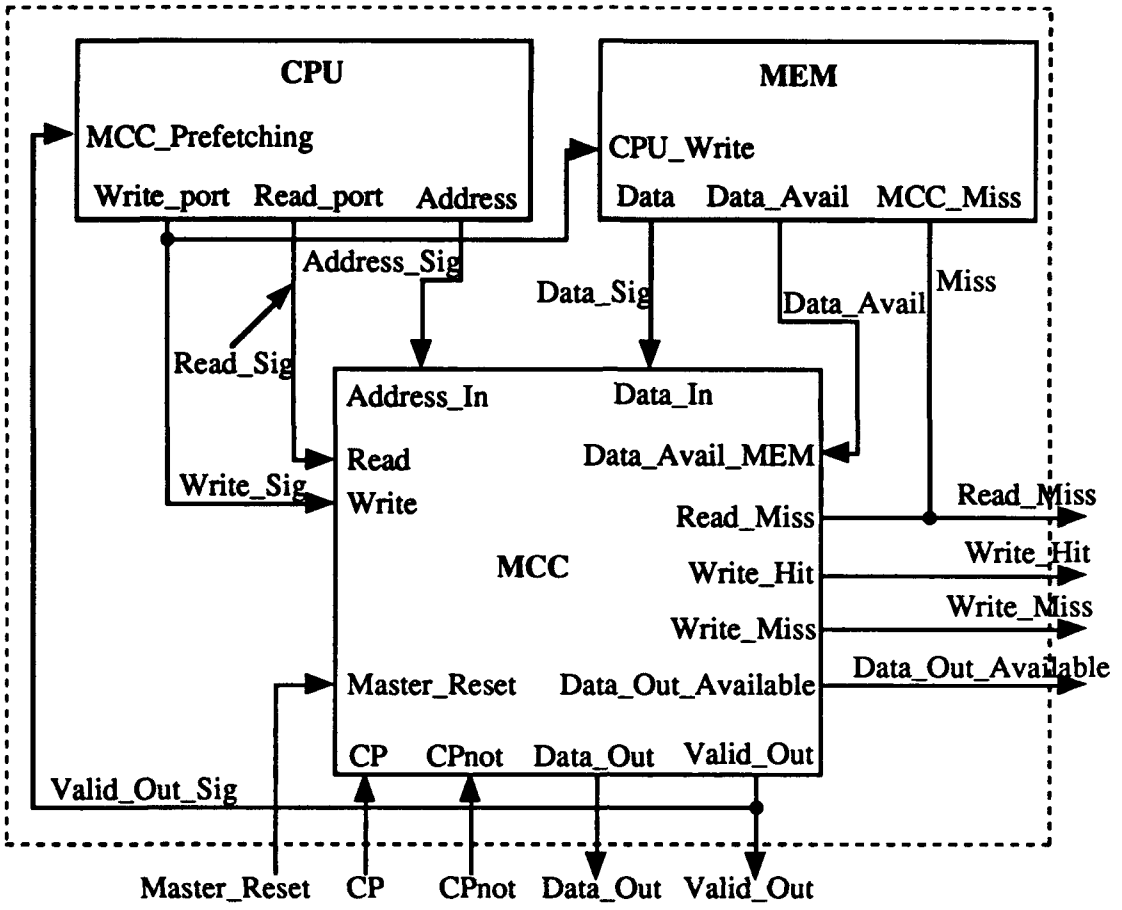


Figure 42. Simple Memory System Used to Test MCC

accomplished. During the prefetch cycle, the MCC's Valid_Out port is asserted high notifying the CPU that no more requests should be generated. This essentially blocks the CPU. If the address is not in the MCC, then the Read Miss state is entered. The Read_Miss port goes high signaling the MEM that it needs to supply data to the data bus. After MEM_Delay, the MEM supplies the data and asserts the Data_Avail signal. The MCC then takes the data off the bus and writes it into the CAM array.

When the CPU requests a write, the signal Write_Sig notifies the MEM to supply data to the data port. The MCC then searches its CAM array for the address. If found,

the Write Hit state is entered and the write is performed. Otherwise, the Write Miss state is entered and the system waits for the next memory request.

Context

The primary purpose of the MCC is to prefetch memory references in the order the CPU previously used them so structural locality can be captured by the SLC chip. Figure 43 shows the context of the MCC in the memory hierarchy and the data flow between the CPU, the SLC, the MCC, and main memory.

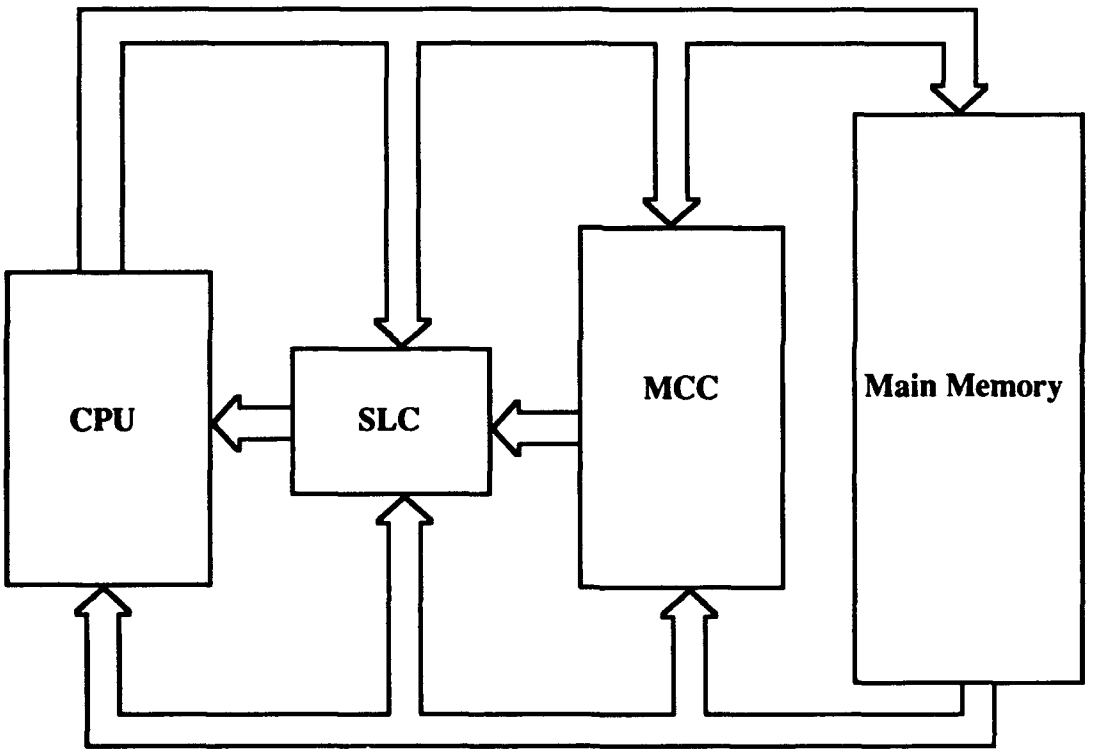


Figure 43. Context of the MCC in the Memory Hierarchy

During a read cycle, the CPU will send a signal signifying that a read is requested, as well as the address of the data to be read. The SLC will accept these data and search its memory for the address. If the address is stored in the SLC, then it, the fastest memory of the hierarchy, will provide the requested data to the CPU. This is the ideal

case and the one that this memory hierarchy is designed for. Since the SLC has captured structural locality, the chances are high that the next reference used by the CPU will also be stored in the SLC.

The MCC also accepts the address from the CPU and does a search in its memory. If the data are found, the prefetch cycle begins and the data are read out of the CAM array in a FIFO manner, one data element per clock cycle. The SLC then stores the data into its memory. One thing to consider is the possibility of a read hit on both the SLC and the MCC. If this occurs, the SLC will be reading data from its memory and providing it to the CPU. Concurrently, the MCC will be in its prefetch mode supplying the SLC with a block of memory references. More research will need to be done to determine what to do if this situation occurs.

One possible solution is simply to ignore the prefetched memory references. This goes along with the assumption that if the SLC contains one reference, then the following references are already contained as structural locality in the SLC. But that will not always be the case. If these references are ignored, then the MCC may be prefetching while another CPU request is generated. If this happens, the MCC may miss a chance to prefetch useful data to the SLC. It may also miss the opportunity to write data to the TOS, thus potentially destroying structural locality. Data may also need to be written into the MCC's CAM array during the prefetch cycle, thus creating cache incoherency in the memory hierarchy. For these reasons, this scheme is not a feasible solution.

Another alternative would be to block the CPU so no other operation can take place on the data buses, except between the MCC and SLC where the prefetching is occurring. If this is done, the CPU will be idle for the number of clock cycles needed to prefetch the entire block of data to the SLC.

The most feasible way to handle this potential problem is to shut down the MCC's prefetch cycle. This can be done by a signal sent from the SLC to the MCC signifying that the SLC contains the data needed by the CPU. This signal can be ORed with the Master_Reset signal in the MCC to selectively initialize desired components. One component that must not be initialized in this case is the TOS_SHIFTER in the WORD_SELECT component of THE_CONTROLLER. This register needs to retain the TOS pointer so memory references can be written into the CAM array in the same order that the CPU uses them. Also, it is important that the contents of the CAM array not be altered.

If the SLC and the MCC do not contain the necessary data, the CPU gets the data from main memory. In this case, the SLC and the MCC wait for main memory to provide the data to the data bus. Main memory will send a signal signifying when the data are available. After receiving this signal, the SLC and the MCC take the data from the data bus and write them into their respective CAM arrays.

The memory subsystem, as dictated by the current design of the MCC, uses write-through to ensure cache coherency. Therefore, all levels of the memory hierarchy are updated on a write-request from the CPU. The SLC will first search its contents for the address to write to. If the address is present, then the data are written into the SLC's memory. Otherwise, the SLC remains idle. The MCC operates the same as the SLC in this case. The main memory performs the conventional write operation.

Performance

This section describes the overall timing for the MCC. The time needed to initialize the MCC and the timing for each of the four states is described. The fastest clock speed allowable for the MCC is 34 ns (29.4 Mhz). This is dictated by the Read Hit state when the MCC uses the clock to prefetch data from the CAM array. Specific time-delays

in the VHDL code were used in the simulation of the MCC and the justification for their use will be explained. The critical paths through the MCC are presented in the next section entitled *Analysis*.

Time for Initializing. Figure 44 shows the timing diagram for initializing the MCC. The signal Master_Reset is used to reset the entire chip model. This signal can be '1' for as little as 38 ns and still initialize the entire MCC. After initialization is complete, Master_Reset must return to '0' before the MCC can perform any operations.

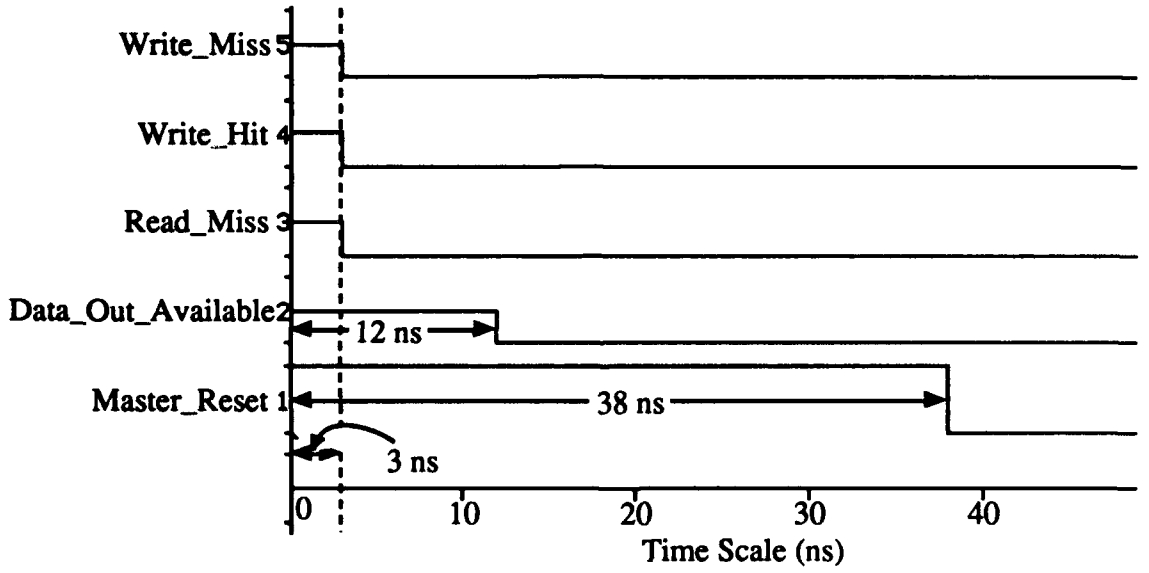


Figure 44. MCC Initialization Timing Diagram

Timing for the Read Hit. Figure 45 shows the timing of the MCC during an entire read hit cycle. Figure 46 is a closer look revealing more detail. Notice that the Data_Out signal transitions at regular intervals. This is due to the synchronous behavior of the MCC during this state. The SHIFT_REGISTER in the WORD_SELECT component of THE_CONTROLLER is shifted upward on the clock pulse, thus allowing the CAM array to perform the read operation.

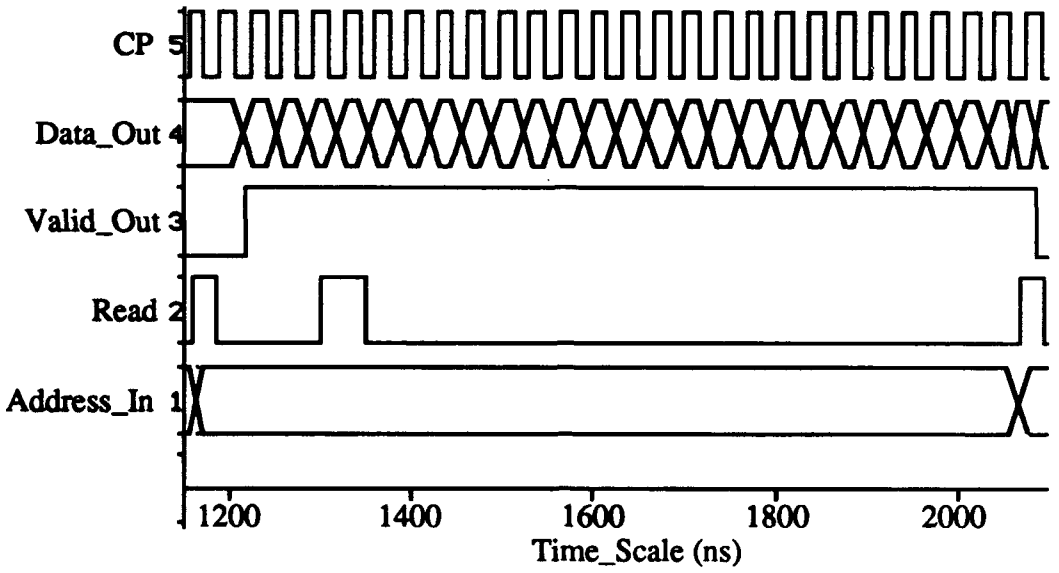


Figure 45. Read Hit Timing Diagram

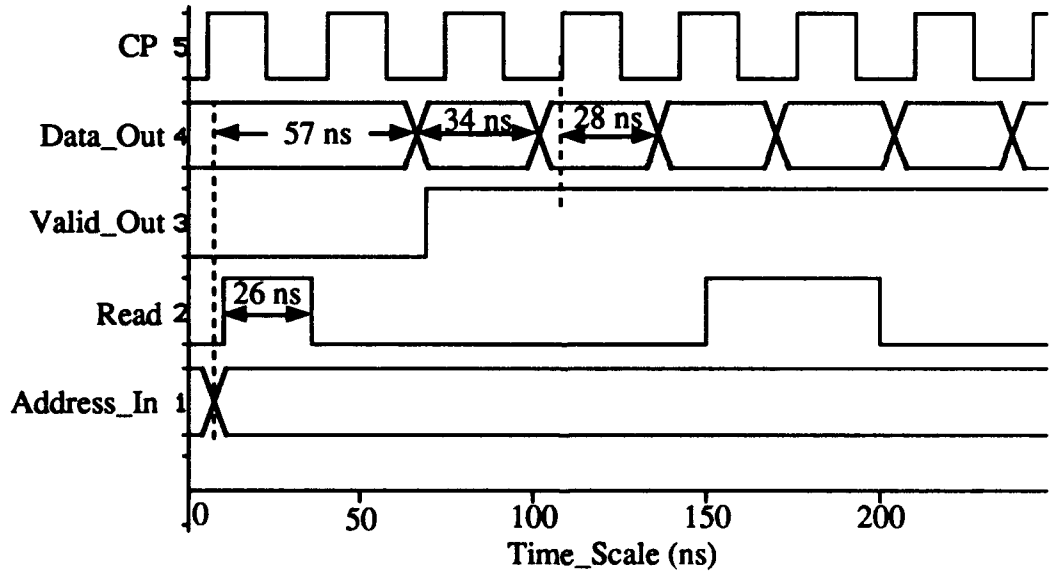


Figure 46. Read Hit Timing Diagram - A Closer View

The data to perform the search for the read function should be on the Address_In ports no longer than 2 ns before the Read port is asserted high. Otherwise, the MCC will not enter the prefetching cycle. This restriction can be relaxed by adding an

Address_Enable port to the MCC. This port would replace the functionality of the CHANGE_DETECTORs in the SEARCH_STATUS component of the CONTROLLER, allowing the address to be on the bus at any time before or after the Read port is asserted high. It takes 57 ns after the Address_In ports are asserted with the address for the first output to be placed on the Data_Out ports. Subsequent data are output onto the Data_Out ports at intervals equal to the clock period. It takes 28 ns after each clock pulse for the data to be placed on the Data_Out ports.

The Read port needs to be high for only 26 ns. After that, the CPU is free to request another operation, although the MCC will not respond because it is prefetching data. This is demonstrated by Figures 45 and 46. From 150 to 200 ns, the Read port is high, simulating a read request by the CPU. The MCC does not respond to this new request because it is in the prefetching cycle.

The Valid_Out signal is asserted high 3 ns after the first data word is available on the Data_Out ports. For each read, the Data_Available signal goes high 7 ns after the data element is available, then transitions to low 5 ns later. After the last valid data element is prefetched and the Data_Available signal goes low, another read is performed on the CAM array and the next word of the array is placed on the Data_Out ports. This time the signal Data_Available does not go high. Therefore, the data should not be read as valid.

At the end of the prefetch cycle, the Data_Out ports become all '0's and 3 ns later the Valid_Out signal goes low. This signifies that any data on the Data_Out ports is not valid.

The total time required for the Read Hit state depends upon when the clock pulse enters the MCC. The CP signal is ANDed with the Counting signal, which in turn clocks the SHIFT_REGISTER. Thus, the time required for this state can vary by as much as

one-half of a clock cycle. Using a clock speed of 34 ns, the range of time would be 917 ns to 934 ns while prefetching 25 memory references.

Timing for the Read Miss. The timing diagram for the signals involved in a Read Miss state is shown in Figure 47. After the Read_Miss signal is asserted high, the MCC must wait for the Data_Avail_MEM signal from main memory to be asserted high before writing the data into the CAM array.

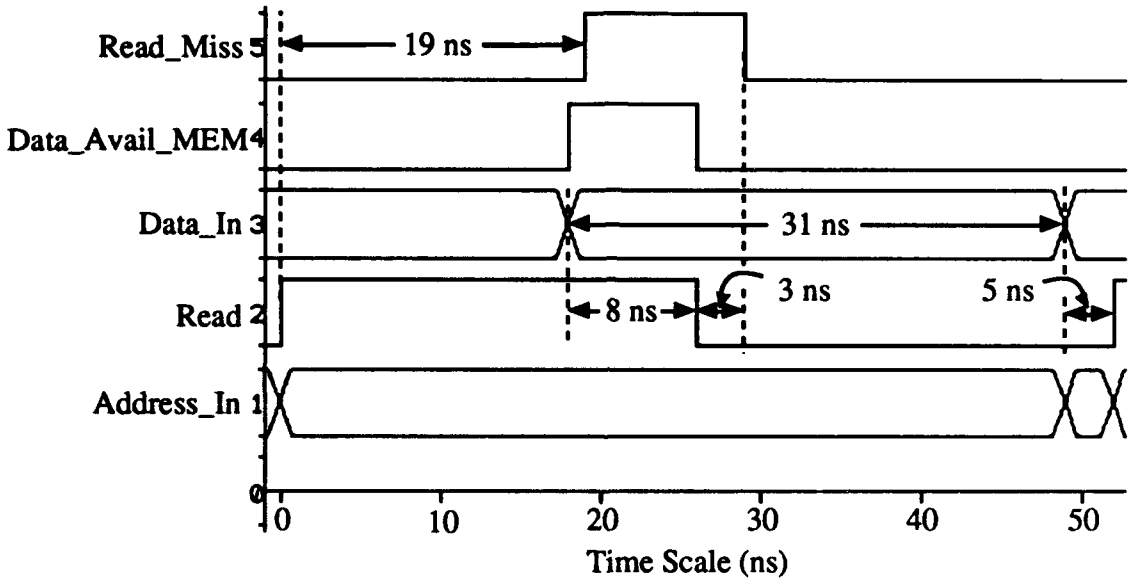


Figure 47. Read Miss Timing Diagram

The Read_Miss port of the MCC is asserted high 19 ns after the Read port goes high, indicating that the requested data are not stored on the MCC. So that the CPU is slowed as little as possible, the MCC allows the CPU to take the Read signal low as soon as 8 ns after the Data_Avail_MEM port goes high. The Read_Miss port will go low 3 ns after the Read port goes low.

When main memory asserts the Data_Avail_MEM port high along with supplying the data onto the data bus, the MCC performs the write function and adds these data to

the TOS of the CAM array. The Data_Avail_MEM port can go high 18 ns after the read is requested. Even though the Read_Miss port is asserted high 19 ns after the read is requested, the MCC is set up to do the write into the CAM array after only 18 ns. This, of course, is not realistic since the memory would be slower than the MCC. The point is, the MCC is set up to do a write into the CAM array with plenty of time to spare. Data_Avail_MEM must be high for at least 8 ns to allow the write to occur. It should not be high any longer than it takes the CPU to request another operation of the MCC. The data should be available for 31 ns after Data_Avail_MEM goes high.

A 5 ns time period must exist after a Read Miss state and the next requested operation. Therefore, the Read Miss state takes a total of 54 ns to complete.

Timing for the Write Hit. Figure 48 displays the timing diagram for the signals involved during the Write Hit state. Here, the CPU supplies the address and data to be written into memory and asserts the Write port high. It takes 31 ns for the Write port transition to reset the SHIFT_REGISTER in the WORD_SELECT component of THE_CONTROLLER. Therefore, this state requires that the CPU take the Write port low at least 28 ns before the address and data are taken off the data buses. The 3 ns difference is accounted for by the AND gates inside each CAM cell that the data must pass through. If the SHIFT_REGISTER is not reset before the address and data are changed, invalid address and data will be written into the CAM array. The Write port can go low as little as 20 ns after transitioning to high.

When a write hit occurs on the MCC, it takes 19 ns for the Write_Hit port to change to '1'. Only 3 ns after the Write port is asserted low will the Write_Hit port transition back to '0'. The Write Miss state requires 48 ns before the next operation can take place.

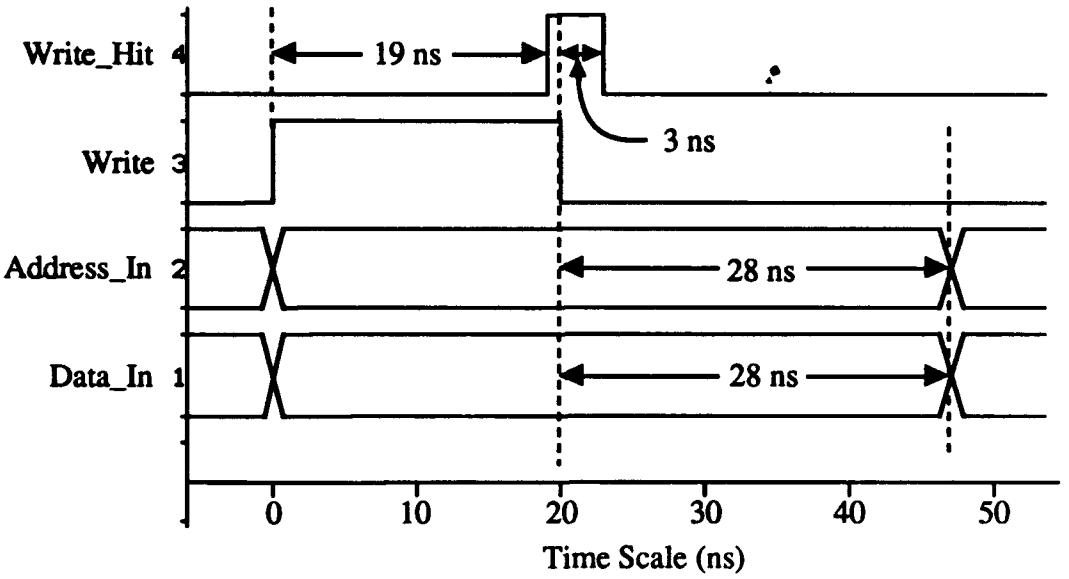


Figure 48. Write Hit Timing Diagram

Timing for the Write Miss. The timing for the participating signals of the Write Miss state is shown in Figure 49. The CPU supplies the same type of data as in the Write Hit state above; namely the address, the data, and the Write signal. The Write port must be high and the address and data must be available for at least 20 ns after the write is requested. When the search does not find the address in the CAM array, the Write_Miss port is asserted high after 19 ns. When the Write port is changed back to '0', it takes only 3 ns for the Write_Miss port to return to '0'. The MCC then needs 24 ns after the Write port goes low to reset itself before another operation can begin. The Write Miss state requires 44 ns to complete.

For both the Write Hit and the Write Miss states, the entire computer system must wait for main memory to write the data into main memory before any other operation can occur. This is a drawback of using a write-through policy with no buffering.

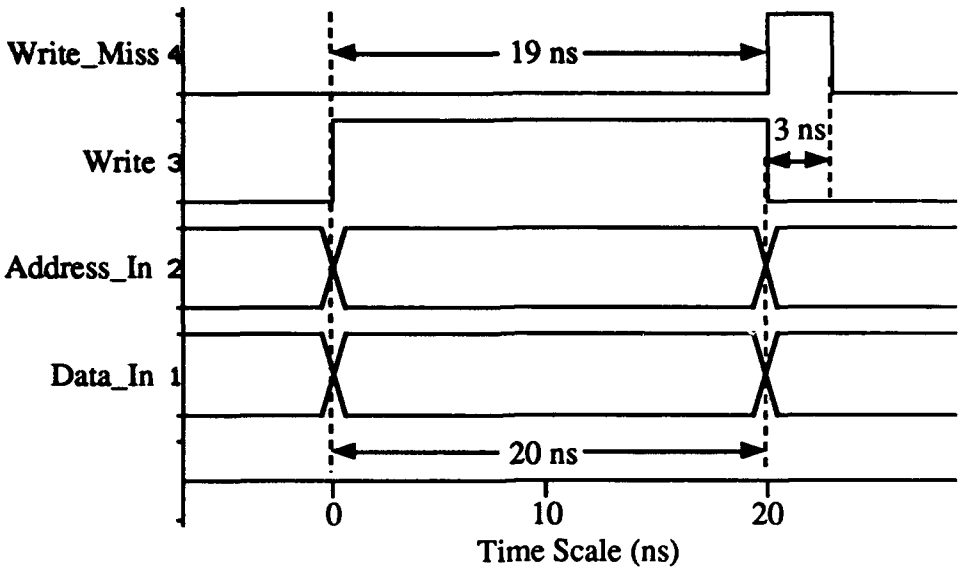


Figure 49. Write Miss Timing Diagram

Timing Justification. The time delays used by the MCC are shown in Table 8. They are defined in chip_pkg.vhd (see Appendix E) and were used as *generics* throughout the entire model. These generic time delays were supplied by Mehalic (14) who says they are figures derived through SPICE simulations and the actual testing of hardware at the Air Force Institute of Technology.

Table 8

Generic Time Delays Used in the MCC

Circuit	Time Delay (ns)
AND Gate	3
Buffer	1
D-type flip-flop	6
Inverter	1
Multiplexer	2
NAND Gate	2
NOR Gate	3
OR Gate	4
XNOR Gate	4
XOR Gate	4

These delay values were used for the rise and fall inertial delays for each component. This will not necessarily be the case for a SPICE model nor for the actual hardware implementation of this circuit. For all of the components, the rise inertial delay will probably be different than its fall inertial delay. The actual delays will also more than likely have decimal values. Due to these considerations, the timing of the fabricated chip will be different than the simulation.

Analysis

We now know the time it takes for each of the states to complete its cycle. Let's now investigate why it takes each state the amount of time it does. The critical paths through the chip model determine this time and can be useful for anyone desiring to speed up the MCC.

Read Hit Analysis. The Read Hit is the longest state simply because it prefetches a block of data when activated. The state can be broken into three separate phases. The first phase is the one in which the MCC is setting up for the prefetch cycle. The second phase is the prefetch cycle when the data words are being read on each clock cycle. The third and final phase is when the PREFETCH_COUNTER has completed its counting to the pre-specified prefetch block size and when the MCC is resetting itself for the next operation.

Figure 50 shows the signals of the critical path during the first phase of a Read Hit cycle. This phase, during any read hit, takes 57 ns finish. It is in this phase that the first data word is read from the CAM array.

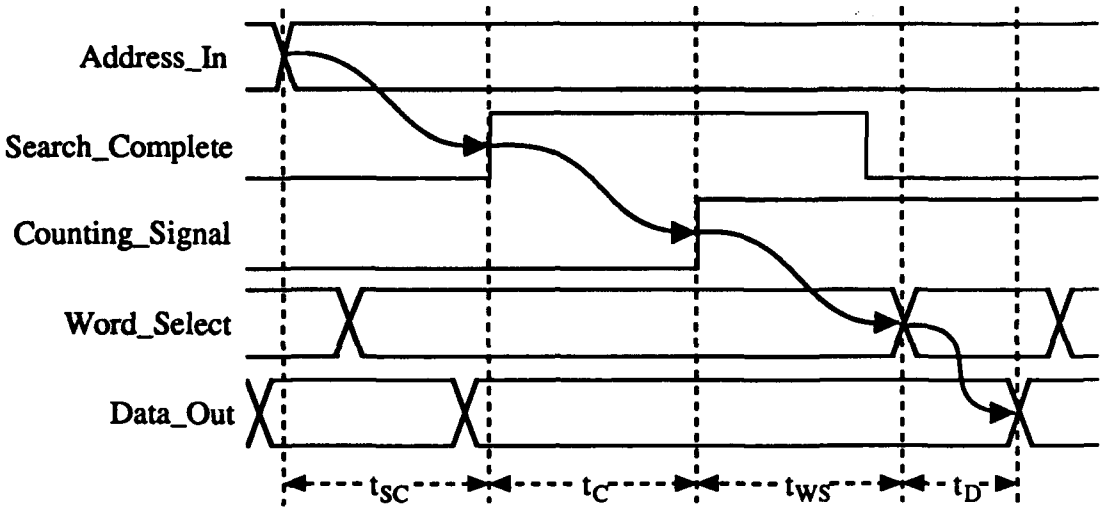


Figure 50. Critical Path Timing Diagram for Read Hit in First Phase

The times used in Figure 50 are defined as follows.

t_{SC}: This is the time it takes the Search_Complete signal to go high after the Address is available to the MCC. The path for this time is through the SEARCH_STATUS component of THE_CONTROLLER.

t_C: This is the time it takes the PREFETCH_STATUS component to output a '1' onto the Counting_Signal after the Search_Complete signal goes high.

t_{WS}: This is the time it takes the WORD_SELECT component to load the SHIFT_REGISTER and output the correct data onto the Word_Select_Bus so a read can be accomplished. Counting_Signal triggers this action.

t_D: This is the time it takes after the Word_Select_Bus gets its data until the output data from the CAM array is on the MCC's Data_Out ports.

The values of these times are given in Table 9. The total time, therefore, for the first phase of the Read Hit is given by

$$t_{RH1} = t_{SC} + t_C + t_{WS} + t_D$$

which calculates to 57 ns.

Table 9

Signal Times for the First Phase of the Read Hit State

	Time (ns)
t_{SC}	16
t_C	16
t_{WS}	16
t_D	9

The second phase of the Read Hit state consists of the data being read on each *clock pulse*. Figure 51 shows the critical path through the MCC during this phase. The figure shows only one of the many data words being read. The data are put onto the Data_Out ports at each clock interval.

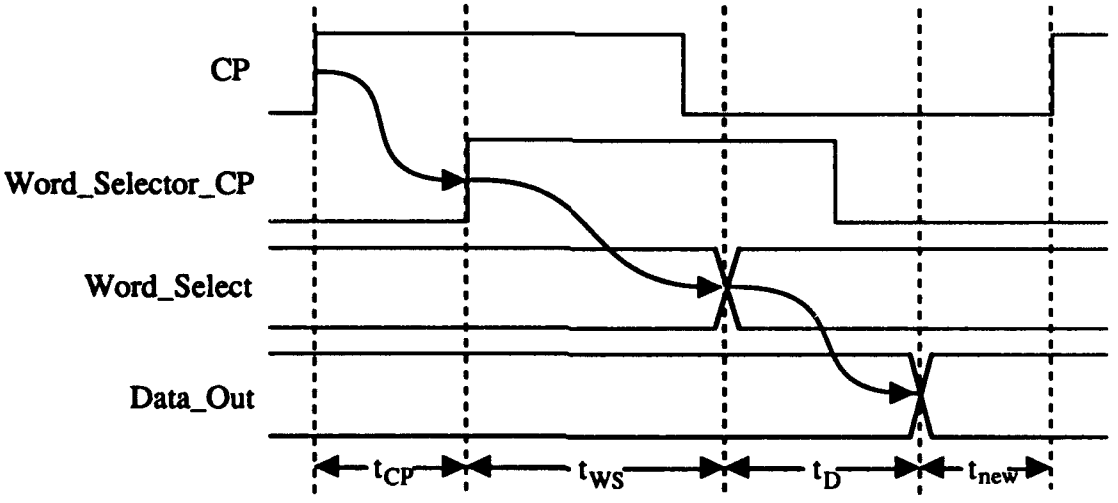


Figure 51. Critical Path Timing Diagram for Read Hit in Second Phase

The times used in Figure 51 are defined as follows.

t_{CP}: This is the time it takes for the clock pulse to traverse the WORD_SELECT_CLOCK component to be input into the WORD_SELECT component.

t_{WS}: This is the time it takes for the Word_Select vector to put its data onto the Word_Select_Bus after the Word_Selector_CP signal is input into the WORD_SELECT component.

t_D: This is the time it takes for the data to be put onto the Data_Out ports of the MCC after the Word_Select_Bus has received the data specifying the word to perform the read on.

t_{new}: This is the time it takes, after the data are on the Data_Out ports, for the clock to go high again to initiate another data word to be read.

The values of these times are given in Table 10. The total time, therefore, for the second phase of the Read Hit is given by

$$t_{RH2} = t_{CP} + t_{WS} + t_D + t_{new}$$

which calculates to 34 ns.

Table 10

Signal Times for the Second Phase of the Read Hit State

	Time (ns)
t_{CP}	7
t_{WS}	12
t_D	9
t_{new}	6

In the third and final phase of the Read Hit state, the MCC outputs all zeros on the Data_Out ports and produces a '0' on the Valid_Out port. This signifies that the Read Hit state is finished and any data on the Data_Out ports should not be considered valid. Figure 52 shows the critical path of this phase.

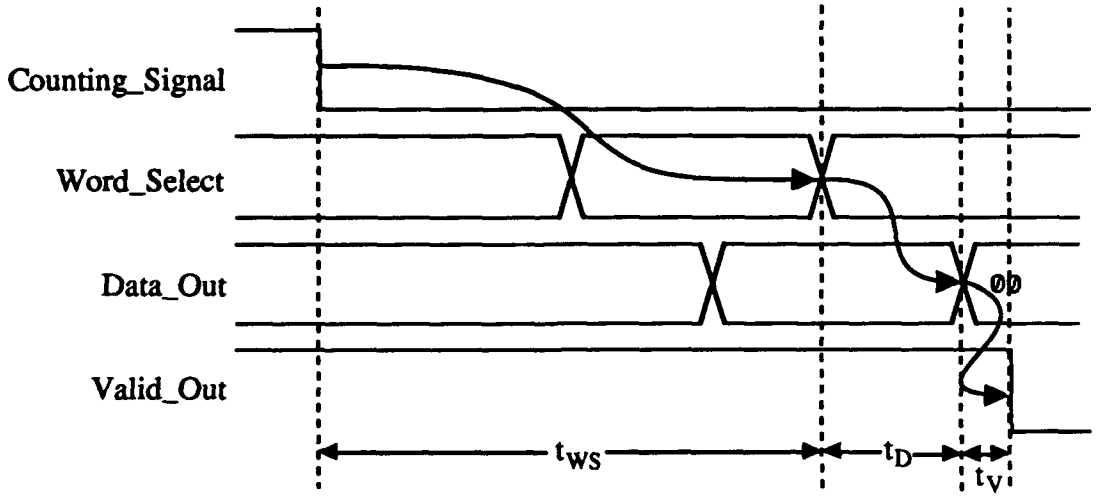


Figure 52. Critical Path Timing Diagram for Read Hit in Third Phase

The times used in Figure 52 are defined as follows.

t_{WS}: This is the time it takes for the WORD_SELECT component to use the Counting_Signal to reset the SHIFT_REGISTER and output all zeros onto the Word_Select_Bus.

t_D: This is the time it takes for the CAM array to output non-valid data and for this data to be put onto the Data_Out ports of the MCC.

t_V: This is the time it takes the MCC to output a '0' on the Valid_Out port after Data_Out becomes all zeros at the end of this state.

The values of these times are given in Table 11. The total time, therefore, for the third phase of the Read Hit is given by

$$t_{RH3} = t_{WS} + t_D + t_V$$

which calculates to 44 ns.

Table 11

Signal Times for the Third Phase of the Read Hit State

	Time (ns)
t_{WS}	32
t_D	9
t_V	3

An entire Read Hit cycle can be as short as 917 ns and as long as 934 ns. During the first phase of this state, the first clock pulse occurs, so in order to calculate the total time of the state we need to multiply t_{RH2} by 24 (assuming a prefetch block size of 25). The total time, therefore, for the Read Hit state, assuming the clock pulses at the same time the Counting_Signal goes high, is given by

$$t_{TOT} = t_{RH1} + 24t_{RH2} + t_{RH3}.$$

This equation calculates the total time for a Read Hit cycle to be 917 ns. But suppose the clock does not pulse at the same time Counting_Signal goes high. In this case, half a clock cycle could pass before Word_Selector_CP is effected. Therefore, 17 ns (assuming a clock speed of 34 ns) must be added to t_{TOT}, making the prefetch cycle last 934 ns.

Read Miss Analysis. The Read Miss state takes a total of 54 ns to complete before any other operation can be performed on the MCC. This is the minimum time required

for this state. Figure 53 shows the signals that make up the critical path for this state. The port Data_Avail_MEM is dependent on the speed of main memory. Figure 53 shows the earliest time Data_Avail_MEM can go high. The MCC will wait as long as necessary for this signal and for the data from main memory before writing into the CAM array.

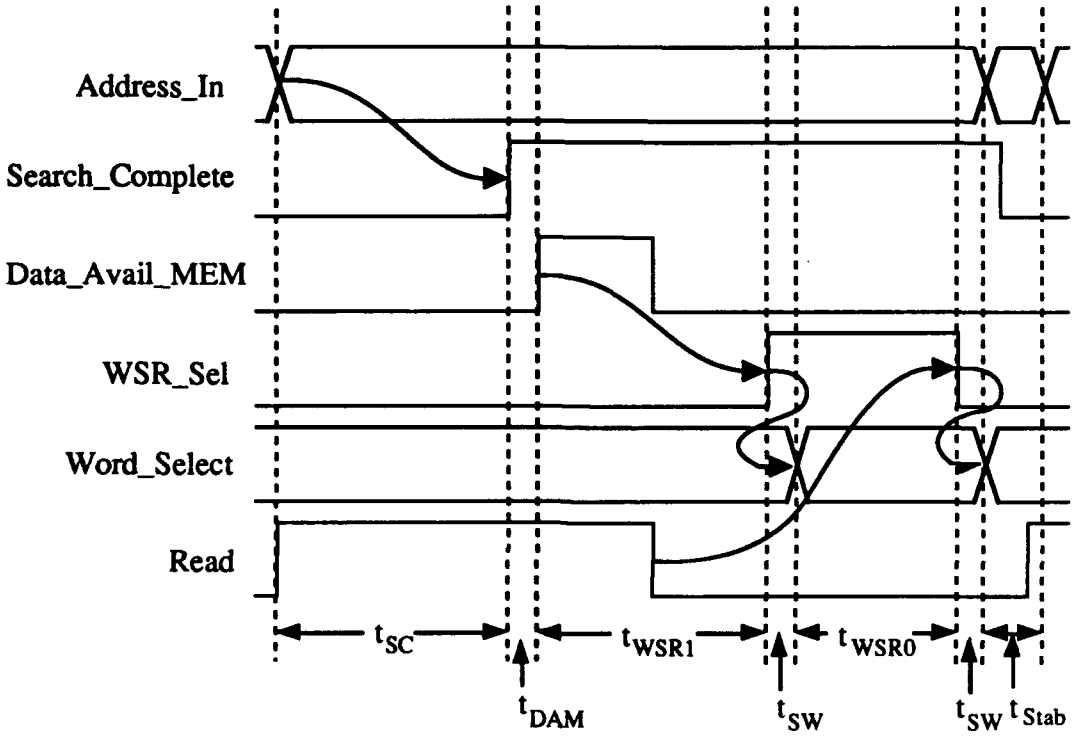


Figure 53. Critical Path Timing Diagram for Read Miss

The signal WSR_Sel is the signal that is input into the multiplexers of the WORD_SELECT component and selects which shift register's outputs will be put onto the Word_Select_Bus. Word_Select is the array of signals that are output from the multiplexers and directly connected to the Word_Select_Bus. The other signals have been discussed earlier. The times used in Figure 53 are described as follows:

t_{SC}: This is the time it takes the Search_Complete signal to go high after the address is available to the MCC. The path for this time is through the SEARCH_STATUS component of THE_CONTROLLER.

t_{DAM}: This is the time between the Search_Complete signal going high and the earliest time the Data_Avail_MEM can be applied.

t_{WSR1}: This is the time it takes the WSR_Sel signal to go high after the Data_Avail_MEM signal goes high. The path for this time is through the SELECT_WORD_SELECT component.

t_{SW}: This is the time it takes for the signals to pass through the multiplexers of the WORD_SELECT component of THE_CONTROLLER.

t_{WSR0}: This is the time between the Word_Select signals changing and the WSR_Sel signal being reset to '0'. WSR_Sel is reset by the Read signal going through the FUNCTION_CHANGE_DETECTOR.

t_{Stab}: This is the time it takes the CAM cells to stabilize after all operations are completed in the Read Miss state.

The values of these times are shown in Table 12. The total time, therefore, for the Read Miss to complete is given by

$$t_{RM} = t_{SC} + t_{DAM} + t_{WSR1} + 2t_{SW} + t_{WSR0}\, t_{Stab}$$

which calculates to 54 ns.

Table 12

Signal Times for the Read Miss State

	Time (ns)
t_{SC}	16
t_{DAM}	2
t_{WSR1}	16
t_{SW}	2
t_{WSR0}	11
t_{Stab}	5

Write Hit Analysis. The Write Hit state requires 47 ns to complete. Figure 54 shows the critical paths during this state. Again, this is a minimum time. The Write signal supplied by the CPU is the deciding factor as to how long the MCC remains in this state once it is entered. Figure 54 shows the earliest time allowed for the Write signal to be taken low. It can stay high for as long as needed provided the correct address and data are applied to the MCC.

The signal Word_Selector_CP is the input signal to the clock port of the SHIFT_REGISTER. The other signals in the figure have previously been discussed. The times used in Figure 54 are described as follows:

t_{SC}: This is the time it takes the Search_Complete signal to go high after the Address is available to the MCC. The path for this time is through the SEARCH_STATUS component of THE_CONTROLLER.

t_{CP}: This is the time it takes for the Word_Selector_CP to go high after the search is complete (i.e., Search_Complete = '1'). The path for this time is through the WORD_SELECT_CLOCK of THE_CONTROLLER.

t_{WS}: This is the time it takes for the Word_Select signal vector to put the correct data onto the Word_Select_Bus for the write operation to occur.

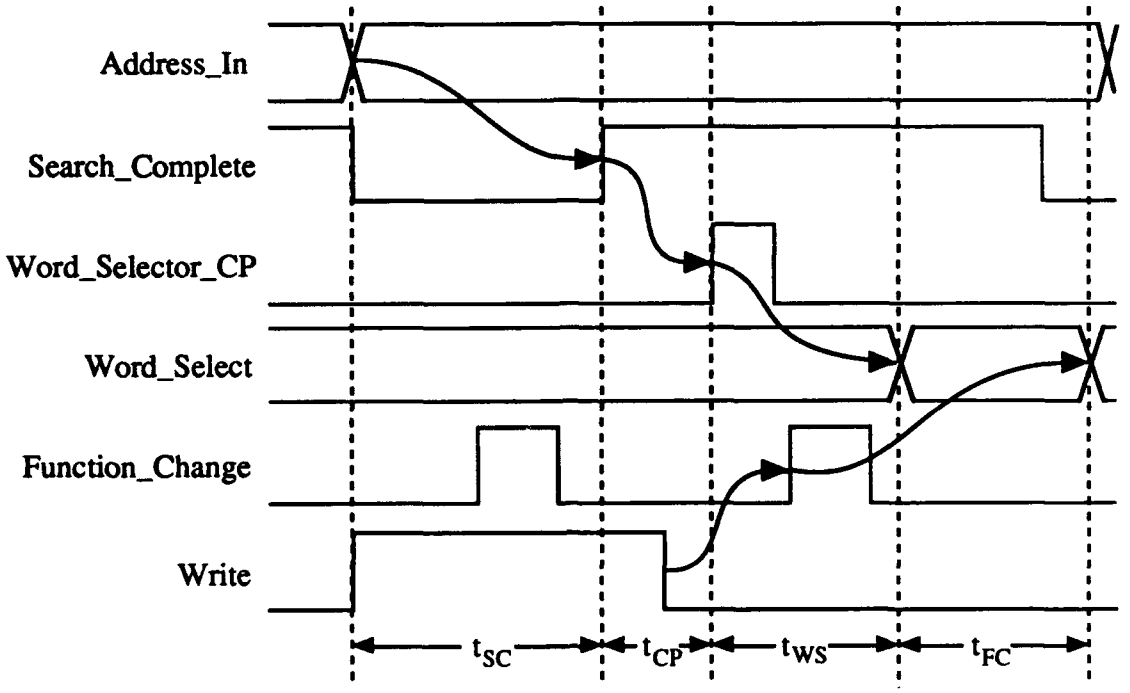


Figure 54. Critical Path Timing Diagram for Write Hit

Word_Selector_CP is the signal that clocks the Resolved_Signal_Tag vector into the SHIFT_REGISTER. The path for this signal is through the WORD_SELECT component of THE_CONTROLLER.

t_{FC}: This is the time it takes after the Word_Select vector puts the data onto the Word_Select_Bus for the write, and before the vector is reset to all zeros. Function_Change is triggered by the change in the Write signal. Function_Change then resets the SHIFT_REGISTER.

The values for these times are given in Table 13. The total time needed for the Write Hit state is given by

$$t_{WH} = t_{SC} + t_{CP} + t_{WS} + t_{FC}$$

which calculates to 47 ns. One nanosecond later the next request can be honored.

Table 13

Signal Times for the Write Hit State

	Time (ns)
t_{SC}	16
t_{CP}	7
t_{WS}	12
t_{FC}	12

Write Miss Analysis. The Write Miss state takes a total of 44 ns to complete. No further action is required of the MCC in this state but it still needs time to perform the search and reset itself for the next operation. Figure 55 shows the critical path through this state. The total time is dependent upon when the CPU takes the Write port low. Shown in Figure 55 is the soonest time the Write port is able to go low without the MCC producing undesired results.

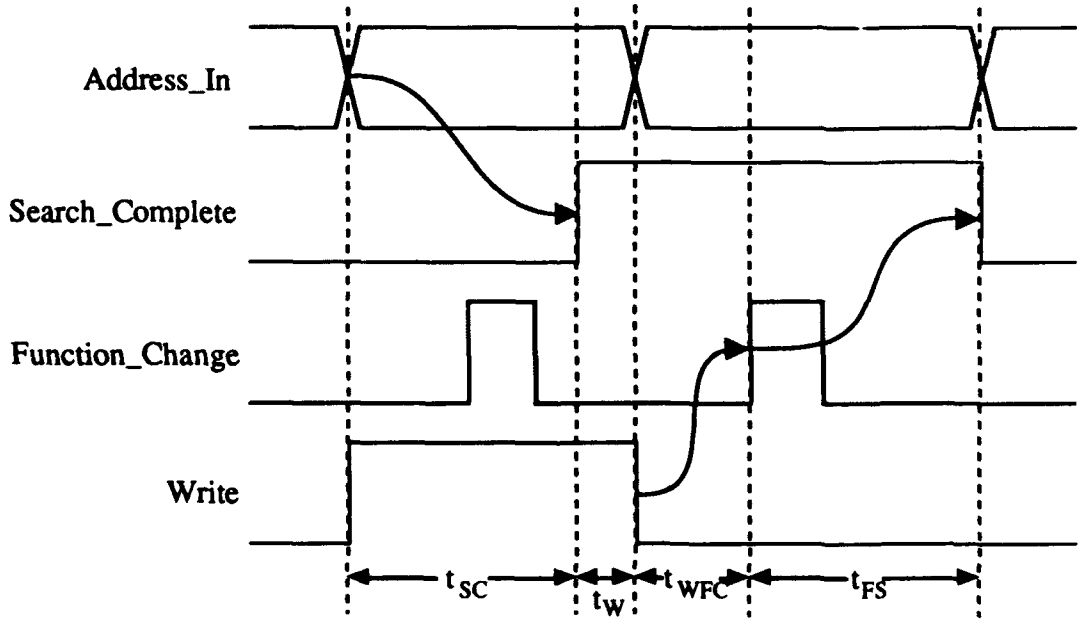


Figure 55. Critical Path Timing Diagram for Write Miss

The times used in Figure 55 are described as follows:

t_{SC}: This is the time it takes the Search_Complete signal to go high after the Address is available to the MCC. The path for this time is through the SEARCH_STATUS component of THE_CONTROLLER.

t_W: This is the time between when the Search_Complete signal goes high and the Write port is able to go low.

t_{WFC}: This is the time it takes the FUNCTION_CHANGE_DETECTOR to output the signal Function_Change after Write goes low.

t_{FS}: This is the time it takes the Function_Change signal to reset the D-type flip-flop in the SEARCH_STATUS component.

The values for these times are given in Table 14. The total time needed for the Write Miss state is given by

$$t_{WM} = t_{SC} + t_W + t_{WFC} + t_{FS}$$

which calculates to 44 ns. At that time the next operation can be performed by the MCC.

Table 14

Signal Times for the Write Hit State

	Time (ns)
t_{SC}	16
t_W	4
t_{WFC}	8
t_{FS}	16

Possible Improvements. One possible improvement for the overall speedup of the memory subsystem is to use a buffered write-through policy. This will allow the CPU to continue its operations without having to wait on main memory to finish a write. This can be accomplished by adding buffer registers to the MCC. Therefore, when the CPU requests a write operation, the MCC will write the data into the buffers. The CPU can then continue working while the MCC waits for the data bus. When the data bus is free, the MCC places the data onto the bus and main memory writes these data into its storage.

A disadvantage to using this policy is the addition of complexity to the MCC. By adding write-through buffers, more control logic will be needed on the chip. This will not only add to the complexity of the chip but it will possibly decrease its storage capacity. The trade-off is speed versus space. If the buffers are used, the CPU can continue its operations without having to wait on main memory. On the other hand, will the decrease in memory storage in the MCC be enough to reduce the amount of structural locality the MCC can hold? Probably not, and since the purpose of this research is to speed up a computer as much as possible, a buffered write-through policy would be advantageous.

The CHANGE_DETECTOR component was used liberally throughout the design to detect a change in a desired signal. The output of the CHANGE_DETECTOR would go high for a short period of time then go low. This period of time was somewhat arbitrarily chosen to be 5 ns. The output needed to be '1' for a short enough period of time not to slow the MCC down too much, yet be longer than the inertial delays of the gates it entered. This period could be shortened to 4 ns since the longest inertial delay of any gate it enters is 4 ns. Table 15 shows the time savings in each of the four states and the component in which the savings could be realized. It is worth noting that the OPERATION_STATUS also contains a CHANGE_DETECTOR but the change would only affect the Data_Out_Available port and not the overall speed of the MCC.

Table 15

CHANGE_DETECTOR Time Savings (ns)

	Read Hit	Read Miss	Write Hit	Write Miss
SEARCH_STATUS	1	1	1	1
WORD_SELECT	3			
FUNCTION_CHANGE_DETECTOR	1	1	1	1
WORD_SELECT_CLOCK		1		
Total Time Savings	5	3	2	2

The main goal of the SEARCH_STATUS component of the CONTROLLER was to allow the CAM array plenty of time to do the initial search before any further operations were performed. By adding an Address_Enable port to the MCC, much of the logic contained in this component can be eliminated. Figure 56 shows a recommended design for the SEARCH_STATUS. The total time savings by implementing this new design is 3 ns during each state of the MCC. Currently, the SEARCH_STATUS component takes 16 ns to complete its operations. In the new design, 13 ns is all the time required: 5 ns through the CHANGE_DETECTOR and 8 ns to accomplish a write into the D-type flip-flop.

The design of the MCC did not consider prefetching data above the current TOS pointer. An obvious way to see the problem this presents is during the initial operations of the MCC in the memory system. If a Read Hit state is entered before the CAM is filled with data, it is possible that bad data above the TOS pointer could be prefetched. If the CAM array is filled with data, a similar situation can occur. Since the TOS pointer is pointing at the top of the stack, any prefetches beyond the pointer retrieve data from the bottom of the stack, thus potentially destroying the prefetch of structural locality. This problem can be fixed by comparing the output of the SHIFT_REGISTER (containing the prefetch pointer) with the output of the TOS_SHIFTER (containing the TOS pointer). When the outputs are the same, the prefetch cycle should be shut down.

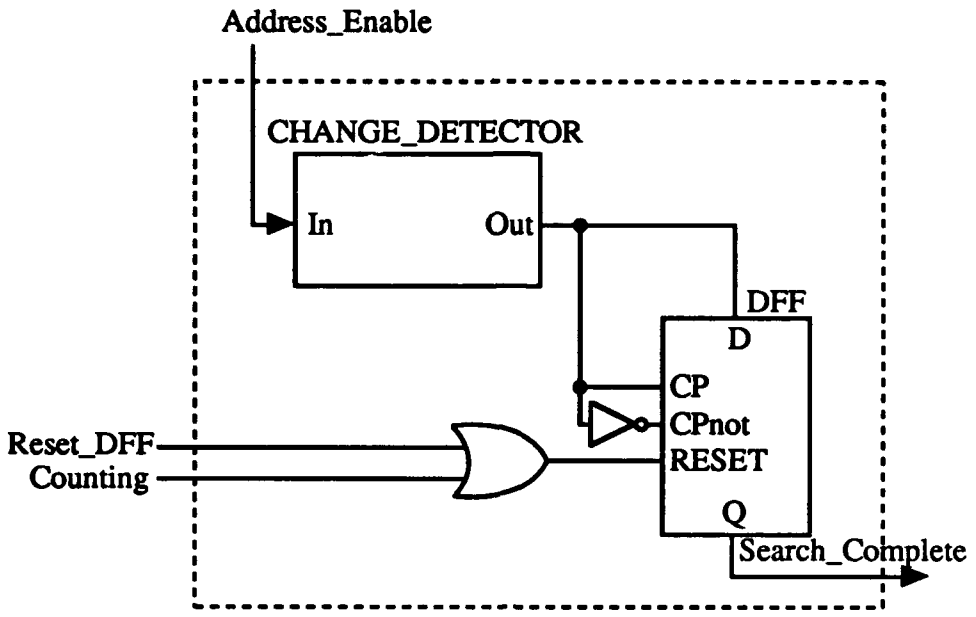


Figure 56. New Design of the SEARCH_STATUS Component

Space Savings. Time was a major factor during the designing and building of the MCC in VHDL that prevented the consideration of space savings on the final hardware chip. In this section, the major areas recommended for redesign are discussed. The resulting decrease in the number of transistors will allow the hardware product to be much smaller than it would be if the current design is used.

The CAM cell of Figure 28 is a primary component to consider for redesign. The cell, as it stands, would require 60 transistors to implement. This assumes implementation in CMOS technology without redesign and the number of transistors for each gate shown in Table 16. In contrast, some of the cells described in Chapter 2 of this thesis require only 5 transistors. This is 8.33% the size of the cell in Figure 28. If a 128 x 64 array of these CAM cells were built into the MCC, a savings of 450,560 transistors would be realized.

Table 16

Transistor Requirements for Gates of Figure 28

Inputs	Gate	Transistors
3	AND	8
	Inverter	2
2	NOR	4
3	NOR	6

The CAM cell used in this thesis has two output signals: RX and RY. RX is the complement of the cell content and RY is the actual value of the cell content. These two signals combined are exclusively-ORed to produce the validity bit. This bit is used during the read operation to determine if the data are valid when a multiple read occurs. Since the MCC writes data into the CAM array only when a read miss or a write hit occurs, the situation with multiple words containing the same address will never happen. Therefore, this port, the Valid_Out port, can be eliminated from the MCC. Consequently, a CAM cell with only one output signal is all that is needed.

Another possible place to save chip area is to replace the CAM cells used to store the data portion of the memory word with DRAM. The data portion of the word is never searched during the operation of the MCC. It is merely a storage device to read from and write to. Therefore, a CAM cell with search capabilities is superfluous in this application. A DRAM can store these data with better space efficiency and still be able to access the data quickly. DRAMs consist of only 1 transistor. Therefore, a DRAM is 1.67% the size of the CAM cell used in this thesis and 20% the size of a 5-transistor CAM cell.

Space can also be conserved by redesigning the PREFETCH_COUNTER component (Figure 24). This component uses a binary counter (Figure 19) to count upwards while comparing the result with a predefined value (the prefetch block size). While the values are not equal, the component outputs a '1' from the RS-type flip-flop onto the

Counting port. When the output of the binary counter equals the prefetch block size, the RS-type flip-flop is reset and the Counting port becomes '0'. If the binary counter were replaced with a count-down counter, the register holding the prefetch block size could be removed and the XNOR gates could be eliminated. The AND1 gate could then be replaced by a NOR gate, so when the count-down counter reached all '0's, the NOR gate would output a '1' and reset the RS-type flip-flop. These changes would reduce the complexity of the circuit and possibly increase its speed.

The new design of SEARCH_STATUS shown in Figure 56 has eliminated much of the required logic in that component. It allows the following gates to be deleted from the current design: two AND gates, two OR gates, and one NOR gate. In addition, only one change detector is needed, which eliminates the need for Address_length-1 XOR gates and buffers.

Two AND gates from THE_CONTROLLER can be eliminated. AND2 of the OPERATION_STATUS component has the same inputs as AND1 of the WORD_SELECT_CLOCK component. Also, AND4 of the OPERATION_STATUS component has the same inputs as AND2 of the WORD_SELECT_CLOCK. Therefore,in each of the two cases, one of the gates can be deleted provided the output of the remaining gate is used to satisfy the requirement of the one deleted. In an hierarchical design such as the design in this thesis, it is not surprising to find this type of duplication.

Hardware Implementation Issues

Some hardware issues could not be considered in this research since only the designing and VHDL implementation of the chip were performed. This section will provide a few areas to investigate before fabrication.

Since the address and data fields are of generic size on the MCC, the total number of pins on the chip is unknown at the writing of this thesis. Aside from the Data_In and

Address_In ports, only 12 other pins are needed. Therefore, a chip made with the data and address being 32 bits wide each, for example, would require 140 pins on the chip (i.e., (32 + 32) x 2 + 12 = 140). These are typical word lengths but if pin-out does present an obstacle, a solution exists to circumvent the problem. A bidirectional buffer to hold the incoming address and data and output the out-going data could be used. This would add to the complexity of the chip but would decrease significantly the number of pins required and would allow for much larger address and data fields.

The number of cells in the CAM array is also not determined in this research. The number of CAM cells a signal can drive before losing its value is finite. Therefore, some form of signal enhancement must be used to allow a signal to propagate to all cells of the CAM array with enough power for each cell to perform the desired operation. The use of buffers is the usual means of performing this task. The location of these buffers must be determined by finding how many cells a signal can drive before losing too much of its value.

During the initialization phase of the MCC, the Master_Reset signal was used to fill the entire CAM array with zeros. This was necessary to avoid operations on *X*s (unknowns) that are present in the CAM array upon startup. The problem with this approach is that there is likely an address in main memory made up of all zeros. So, if the CPU requests an operation on that address before the CAM array is filled with valid data, bad data could possibly be sent to the CPU from the MCC. As mentioned previously, a search is the first operation to be performed when the MCC is activated. A search on an *X* produces *X*s on the tag lines, which is not acceptable in the completion of the state operations. A cell that can effectively use an *X* and output a '0' on the tag line during a search would be ideal. Another solution would be to resolve *X*s to '0's on a resolved line. Before implementing this chip in hardware, this issue must be resolved.

Noise was a concern with the AT&T WE-32201 Integrated Memory Management Unit/Data Cache (IMDC). This chip is used to translate virtual addresses to physical addresses. A large noise spike can occur during a normal translation when many cells in the CAM discharge at once. The designers of this chip used very wide power and ground buses and multiple V_{SS} tub ties to keep the noise under 400mV. Therefore, noise should be considered during the implementation phase of the CAM chip described in this thesis. (5:595)

Summary

This chapter provided an in-depth analysis of the MCC. The complete functionality of the MCC was presented as well as the context of the MCC in a memory hierarchy. The performance of the chip model was carefully analyzed and it was determined that the Read Hit state took the longest to complete for an obvious reason: this is the state in which the prefetching of memory references occurs. The Read Hit state takes 57 ns + (Prefetch_Block_Size-1)(Clock_Period) + 44 ns to complete. During the testing of the MCC, the Prefetch_Block_Size was set at 25 and the Clock_Period to 34 ns, making the total time to complete the prefetch cycle 917 ns to 934 ns. The remainder of the states, the Read Miss, the Write Hit, and the Write Miss, take 54 ns, 48 ns, and 44 ns, respectively, to complete. The critical paths through these states were determined so the slowest components could be scrutinized for possible speedup. Some suggestions to increase the speed of the MCC were presented as well as potential areas for saving space on the chip. Finally, some hardware considerations were presented for further investigation before the chip is actually fabricated.

V. Conclusions and Recommendations

Introduction

The focus of this thesis has been a proof-of-concept on modeling a cache chip that stores memory references in the order they were used, and prefetching these locations in a FIFO manner so structural locality can be captured by a faster on-chip cache. A content-addressable memory was designed with this in mind. All of the functionalities of the main CAM cache (MCC) were accomplished and a working structural-level VHDL design of the chip was completed.

This thesis is the first iteration of research into the hardware realization of a memory hierarchy that exploits structural locality of memory references. The effort has shown that a main CAM cache that exploits structural locality is a feasible design. The product of this thesis was a VHDL design, starting at the gate level, of a CAM cache that prefetches memory locations in the order they were used by the CPU so an on-chip cache can capture structural locality and provide it to the CPU for fast processing.

Conclusions

The MCC was designed with a bottom-up approach. The functionality of the MCC was first determined; then the logic to implement the functionality was developed. The first operation the MCC performs when activated for any purpose is the search operation. After the search is complete, the MCC enters one of four states: Read Hit, Read Miss, Write Hit, or Write Miss. If the functionality of the MCC were any more complicated than this, then a bottom-up approach would not have been a good one and it may not have been possible to complete the design in one thesis cycle. The approach taken was more intuitive to this researcher. Even if a top-down approach were used, the time it would have taken to incrementally break down a top-level behavioral description into a

structural model would have come close to the time it took to accomplish the bottom-up design.

The functionality concepts of the MCC, i.e., the behavior of the MCC during each of the four states, were successfully implemented in the VHDL model of the MCC. During the Read Hit state, the MCC successfully prefetches the specified prefetch block size of addresses and data allowing another chip, the SLC chip, to write the data into its memory. Thus, structural locality is captured by the SLC. The MCC also writes data into its CAM array in the Read Miss state and writes over old data in the Write Hit state.

Recommendations

The MCC was designed and modeled using the VHDL hardware description language. Before the chip is actually fabricated and placed into a computer memory system, a few areas require further research.

In order for the memory subsystem to be complete, the SLC must be designed. The design will be very similar to that of the MCC. The SLC would have three active states: the Read Hit, Read Miss, and Write Hit states. During the Read Hit state, the SLC would read the requested data from its memory and provide it on its output ports. In the Read Miss state, the SLC would wait on main memory for the data and then write the data into its CAM array using a least-recently-used algorithm The Write Hit state would act similar to, if not exactly like, that of the MCC. The Write Miss state would be an idle state just as it is in the MCC.

The fabrication of the MCC and the SLC would be the next logical step in building this memory subsystem. The size of the chip now becomes a factor. A CAM cell with as few transistors as possible and with the least power dissipation is recommended. Jones (10) presents some excellent arguments as to the selection of a CAM cell with these

two issues in mind. Once the cell is selected and designed, the size of the CAM array can be determined. From there, the MCC and the SLC can be fabricated.

Once the fabrication of the chips is complete, the computer system utilizing this memory subsystem can be bread-boarded and tested. The communications between the components on the bread-board will be similar to the concept shown in Figure 43. If a specially manufactured CPU with on-chip cache having the capability to communicate with the MCC could be produced, the need for the SLC residing off chip could be eliminated. This would allow for quicker on-chip responses rather than the off-chip communication delays.

Summary

The purpose of this type of research is to make computers run faster than ever. By exploiting structural locality, this goal can be reached. This thesis effort produced a design of a CAM cache that stores memory references in the order they were used by the CPU. The cache then provides these data, in a predetermined block size, to a faster cache (this cache captures the structural locality), which in turn provides the data to the CPU. The CPU can then access these data quickly.

Appendix A: The VHDL Code of the CAM Cell

This appendix contains the VHDL code of the CAM cell. It is the heart of the main CAM cache (MCC) chip. The entity is presented first, then the structure begins on a separate page.

CAM_cell Entity

```
--------------------------------------------------------------------------
-- Date:    3 May 1991
-- Version: 1.0
--
-- Filename: cam_cell_entity.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity of the CAM cell.
-- Associated files: cam_cell_structure.vhd : This file contains the gate level
--                                            design of the CAM cell.
--                   chip_pkg.vhd           : This file is where the size of
--                                            the CAM array is defined. Other
--                                            declarations are also contained
--                                            in this file.
--                   chip_pkg_body.vhd      : This file contains the sub-
--                                            routines Wired_And and Wired_Or
--                                            used by the chip.
--                   cam_chip_entity.vhd    : This file contains the entity
--                                            description of the CAM chip.
--                   cam_chip_structure.vhd : This file contains the structure
--                                            of the CAM chip. It is formed by
--                                            generating copies of the CAM
--                                            cell.
--                   chip_stimulus.vhd      : This file exercises the chip and
--                                            provides inputs to test the chip.
--                   chip_test_bench.vhd    : This file contains the test bench
--                                            for the CAM chip.
--                   chip_config.vhd        : This file contains the
--                                            configuration of the system.
--                   clock.vhd              : You guess!
-- History:
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
```

```
entity CAM_cell is
        port(
                D  : in  MVL7;          -- data line into cell
                B  : in  MVL7;          -- bit select line
                W  : in  MVL7;          -- word select line
                M  : out MVL7;          -- match line
                RY: out MVL7;           -- data output (W and C)
                RX: out MVL7);          -- data output (W and Cnot)
                                        -- RY and RX determine the validity of the bit
end CAM_cell;
```

CAM _cell Structure

```
--------------------------------------------------------------------------
-- Date:    3 April 1991
-- Version: 2.0
--
-- Filename: cam_cell_structure.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the architecture structure of the CAM cell.
-- Associated files: cam_cell_entity.vhd     : This file contains the entity
--                                             description of the CAM cell.
--                   chip_pkg.vhd            : This file is where the size of
--                                             the CAM array is defined. Other
--                                             declarations are also contained
--                                             in this file.
--                   chip_pkg_body.vhd       : This file contains the sub-
--                                             routines Wired_And and Wired_Or
--                                             used by the chip.
--                   cam_chip_entity.vhd     : This file contains the entity
--                                             description of the CAM chip.
--                   cam_chip_structure.vhd : This file contains the structure
--                                             of the CAM chip. It is formed by
--                                             generating copies of the CAM
--                                             cell.
--                   chip_stimulus.vhd       : This file exercises the chip and
--                                             provides inputs to test the chip.
--                   chip_test_bench.vhd     : This file contains the test bench
--                                             for the CAM chip.
--                   chip_config.vhd         : This file contains the
--                                             configuration of the system.
--                   clock.vhd               : You guess!
--
-- History: Version 1.0 - used my own gates which were described behaviorally.
--          Version 2.0 (3 April 1991) - switched to ZYCAD components.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use ZYCAD.components.all;
use WORK.Chip_pkg.all;

architecture Structure of CAM_cell is

-- The following are internal signals of the cell
        signal T1:    MVL7;
        signal T2:    MVL7;
        signal C:     MVL7;
```

```
        signal Cnot:  MVL7;
        signal A1:    MVL7;
        signal A2:    MVL7;
        signal Dnot:  MVL7;
        signal Wnot: MVL7;
        signal Bnot:  MVL7;


-- These components make up the CAM cell

        component INVGATE                       -- ZYCAD component
                generic (tLH: Time;             -- rise inertial delay
                         tHL: Time);            -- fall inertial delay
                port(input: in MVL7;            -- one input
                     output: out MVL7);         -- one output
        end component;

        component ANDGATE                       -- ZYCAD component
                generic (N: Positive;           -- N input AND gate
                         tLH: Time;             -- rise inertial delay
                         tHL: Time);            -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);-- N inputs
                     output: out MVL7);         -- one output
        end component;

        component NORGATE                       -- ZYCAD component
                generic (N: Positive;           -- N input NOR gate
                         tLH: Time;             -- rise inertial delay
                         tHL: Time);            -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);-- N inputs
                     output: out MVL7);         -- one output
        end component;

begin

-- component instantiation
        INV1: INVGATE
                generic map (Inverter_Delay, Inverter_Delay)
                                                -- rise inertial delay,
                                                -- fall inertal delay
                port map(
                        input  => D,
                        output => Dnot);

        INV2: INVGATE
                generic map (Inverter_Delay, Inverter_Delay)
                                                -- rise inertial delay,
                                                -- fall inertal delay
                port map(
                        input  => B,
                        output => Bnot);
```

```
INV3: INVGATE
        generic map (Inverter_Delay, Inverter_Delay)
                                        -- rise inertial delay,
                                        -- fall inertal delay
        port map(
                input  => W,
                output => Wnot);

AND1: ANDGATE
        generic map (3,                 -- 3 inputs,
                And_Delay,              -- rise inertial delay,
                And_Delay)              -- fall inertal delay
        port map(
                input(1) => D,
                input(2) => B,
                input(3) => W,
                output   => A1);

AND2: ANDGATE
        generic map (3,                 -- 3 inputs,
                And_Delay,              -- rise inertial delay,
                And_Delay)              -- fall inertal delay
        port map(
                input(1) => Dnot,
                input(2) => B,
                input(3) => W,
                output   => A2);

NOR1: NORGATE
        generic map (3,                 -- 3 inputs,
                NOR_Delay,              -- rise inertial delay,
                NOR_Delay)              -- fall inertal delay
        port map(
                input(1) => D,
                input(2) => Bnot,
                input(3) => Cnot,
                output   => T1);

NOR2: NORGATE
        generic map (3,                 -- 3 inputs,
                NOR_Delay,              -- rise inertial delay,
                NOR_Delay)              -- fall inertal delay
        port map(
                input(1) => Dnot,
                input(2) => Bnot,
                input(3) => C,
                output   => T2);
```

```
NOR3: NORGATE
        generic map (2,                 -- 2 inputs,
                NOR_Delay,              -- rise inertial delay,
                NOR_Delay)              -- fall inertal delay
        port map(
                input(1) => A1,
                input(2) => C,
                output   => Cnot);


NOR4: NORGATE
        generic map (2,                 -- 2 inputs,
                NOR_Delay,              -- rise inertial delay,
                NOR_Delay)              -- fall inertal delay
        port map(
                input(1) => Cnot,
                input(2) => A2,
                output   => C);

NOR5: NORGATE
        generic map (3,                 -- 3 inputs,
                NOR_Delay,              -- rise inertial delay,
                NOR_Delay)              -- fall inertal delay
        port map(
                input(1) => T1,
                input(2) => W,
                input(3) => T2,
                output   => M);

NOR6: NORGATE
        generic map (3,                 -- 3 inputs,
                NOR_Delay,              -- rise inertial delay,
                NOR_Delay)              -- fall inertal delay
        port map(
                input(1) => B,
                input(2) => Cnot,
                input(3) => Wnot,
                output   => RY);

NOR7: NORGATE
        generic map (3,                 -- 3 inputs,
                NOR_Delay,              -- rise inertial delay,
                NOR_Delay)              -- fall inertal delay
        port map(
                input(1) => B,
                input(2) => C,
                input(3) => Wnot,
                output   => RX);

end Structure;
```

Appendix B: The VHDL Code for the Basic Components of the Main CAM Cache

This appendix contains the basic components that make up the MCC. They are listed in alphabetical order and begin on separate pages.

BINARY_COUNTER

```
--------------------------------------------------------------------------
-- Date:    25 July 91
-- Version:  1.0
--
-- Filename:  binary_counter.vhd
-- System:    ZYCAD, VLSI net
-- Language:  VHDL
--
-- Description:  This file contains the entity and structure of an 8-bit
--               binary counter.  This structure was taken from Mano p 278.
--               The outputs are valid on the clock pulse.  Intermediate
--               values may not be valid due to timing inside the JK FFs.
--
-- Associated files:
--       chip_pkg.vhd : This file contains constants, variables, etc. needed
--                      for this file.
--
-- History:
--
-- Author:  Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


entity BINARY_COUNTER is
        port  (Count_Enable  : in MVL7;
               CP            : in MVL7;
               CPnot         : in MVL7;
               CLEAR         : in MVL7;
               Output        : inout MVL7_Vector(Bits_in_Counter-1 downto 0));
end BINARY_COUNTER;


architecture structure of BINARY_COUNTER is
```

```
--------------------------------------------------------------------------
-- This component is a master-slave JK flip flop.
--------------------------------------------------------------------------
        component MS_JKFF
                port(   J       : in MVL7;              -- J input
                        K       : in MVL7;              -- K input
                        RESET: in MVL7;                 -- resets FF to '0'
                        CP      : in MVL7;              -- clock
                        CPnot  : in MVL7;               -- clock complement
                        Q       : inout MVL7;           -- output
                        Qnot    : inout MVL7);          -- output complement
        end component;


--------------------------------------------------------------------------
-- This component is the AND gate.
--------------------------------------------------------------------------
        component ANDGATE                               -- ZYCAD component
                generic (N: Positive;                   -- N input AND gate
                        tLH: Time;                      -- rise inertial delay
                        tHL: Time);                     -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);    -- N inputs
                        output: out MVL7);              -- one output
        end component;


--------------------------------------------------------------------------
-- This component is the Inverter gate.
--------------------------------------------------------------------------
        component INVGATE                               -- ZYCAD component
                generic (tLH: Time;                     -- rise inertial delay
                        tHL: Time);                     -- fall inertial delay
                port(input: in MVL7;                    -- input
                        output: out MVL7);              -- output
        end component;


        signal JK_in: MVL7_Vector(Bits_in_Counter-1 downto 1);
        signal TEMP : MVL7;                     -- output of JK FF for troubleshooting

begin


--------------------------------------------------------------------------
-- The following code generates JK flip flops for the desired number of
--  bits in the binary counter.
--------------------------------------------------------------------------
        JK1:
        for I in Bits_in_Counter-1 downto 0 generate
```

```
        ------------------------------------------------------------
        -- This JK FF holds the least significant bit and the inputs
        -- to this FF come from the Count_Enable line.
        ------------------------------------------------------------
        JK2:
        if I = 0 generate
                JKFF0: MS_JKFF
                        port map(J      => Count_Enable,
                                 K      => Count_Enable,
                                 RESET  => CLEAR,
                                 CP     => CP,
                                 CPnot  => CPnot,
                                 Q      => Output(0),
                                 Qnot   => Open);
        end generate;


        ------------------------------------------------------------
        -- These JK FFs are the rest of the FFs that make up the
        -- counter.
        ------------------------------------------------------------
        JK3:
        if I /= 0 generate
                JK_not0: MS_JKFF
                        port map(J      => JK_in(I),
                                 K      => JK_in(I),
                                 RESET  => CLEAR,
                                 CP     => CP,
                                 CPnot  => CPnot,
                                 Q      => Output(I),
                                 Qnot   => Open);
        end generate;

end generate;


------------------------------------------------------------------------
-- This instantiation is for troubleshooting only. It is a copy of one of
-- the JK FFs above. DO NOT IMPLEMENT IN HARDWARE!!!
------------------------------------------------------------------------
                JKFF: MS_JKFF
                        port map(J      => Count_Enable,
                                 K      => Count_Enable,
                                 RESET  => CLEAR,
                                 CP     => CP,
                                 CPnot  => CPnot,
                                 Q      => TEMP,
                                 Qnot   => Open);
```

```
-------------------------------------------------------------------
-- The following code generates the AND gates in the counter.
-------------------------------------------------------------------
        A1:
        for J in Bits_in_Counter-1 downto 1 generate

                ---------------------------------------------------
                -- This AND gate is the first AND gate. The input Count_
                --  Enable is unique to this gate.
                ---------------------------------------------------
                A2:
                if J = 1 generate

                        AND1: ANDGATE
                                generic map(2,                  -- 2 inputs
                                        AND_Delay,              -- rise inertial delay
                                        AND_Delay)              -- fall inertial delay
                                port map(
                                        input(1) => Output(J-1),
                                        input(2) => Count_Enable,
                                        output   => JK_in(1));
                end generate;

                ---------------------------------------------------
                -- These are generated for the rest of the AND gates.
                ---------------------------------------------------
                A3:
                if J /= 1 generate
                        ANDs: ANDGATE
                                generic map(2,                  -- 2 inputs
                                        AND_Delay,              -- rise inertial delay
                                        AND_Delay)              -- fall inertial delay
                                port map(
                                        input(1) => Output(J-1),
                                        input(2) => JK_in(J-1),
                                        output   => JK_in(J));
                end generate;

        end generate;

end structure;
```

CHANGE_DETECTOR

```
--------------------------------------------------------------------------
-- Date:    12 July 91
-- Version:  1.0
--
-- Filename:  change_detector.vhd
-- System:    ZYCAD, VLSI net
-- Language:  VHDL
--
-- Description:  This file contains the entity entity and structure of the
--               change detector. The component will detect a change in any
--               signal that is input into it. The output will be a '1' for
--               the time it takes the signal to go through the buffer.
-- Associated files:
-- History:
--
-- Author:  Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use ZYCAD.components.all;
use WORK.chip_pkg.all;

entity CHANGE_DETECTOR is
        generic (Delay:  Time);
        port    (Input : in  MVL7;
                 Output: out MVL7);
end CHANGE_DETECTOR;

architecture structure of CHANGE_DETECTOR is

        signal BUF_Out :  MVL7 := '0';

        component BUFGATE
                generic (tLH: Time;                     -- rise inertial delay
                         tHL: Time);                    -- fall inertial delay
                port(input: in MVL7;                    -- one input
                     output: out MVL7 := '0');          -- one output
        end component;

        component XORGATE                               -- ZYCAD component
                generic (N: Positive;                   -- N input XOR gate
                         tLH: Time;                     -- rise inertial delay
                         tHL: Time);                    -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);    -- N inputs
                         output: out MVL7);             -- one output
        end component;
```

```
begin

        BUF1: BUFGATE
                generic map(Change_Detector_Delay, Change_Detector_Delay)
                port map  (input  => Input,
                           output => BUF_Out);

        XOR1: XORGATE
                generic map(2, XOR_Delay, XOR_Delay)
                port map  (input(1) => BUF_Out,
                           input(2) => Input,
                           output   => Output);


end structure;
```

EDGE_TRIGGERED_DFF

```
--------------------------------------------------------------------------
-- Date:    20 August 1991
-- Version: 1.2
--
-- Filename: edge_triggered_DFF.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of an edge-
--              triggered D FF.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc. needed
--                     for this file.
--
-- History: Version 1.0 (30 July 91)
--          Version 1.1 (19 August 1991) - Changed output ports from inout
--              to out and added internal signals to compensate.
--          Version 1.2 (20 August 1991) - added additional port CPnot to
--              aid in resetting the DFF.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;

entity EDGE_TRIGGERED_DFF is
        port(   D       : in MVL7;
                RESET   : in MVL7;
                CP      : in MVL7;
                CPnot   : in MVL7;
                Q       : out MVL7;
                Qnot    : out MVL7);
end EDGE_TRIGGERED_DFF;

architecture structure of EDGE_TRIGGERED_DFF is

--------------------------------------------------------------------------
-- This component is the NAND gate.
--------------------------------------------------------------------------
        component NANDGATE                              -- ZYCAD component
                generic (N: Positive;                   -- N input NAND gate
                         tLH: Time;                     -- rise inertial delay
                         tHL: Time);                    -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);    -- N inputs
                     output: out MVL7);                 -- one output
        end component;
```

```
--------------------------------------------------------------------------
-- This component is the Inverter gate.
--------------------------------------------------------------------------
        component INVGATE                       -- ZYCAD component
                generic (tLH: Time;             -- rise inertial delay
                         tHL: Time);            -- fall inertial delay
                port(input: in MVL7;            -- N inputs
                     output: out MVL7);         -- one output
        end component;


        signal R          : MVL7;
        signal Rnot       : MVL7;
        signal S          : MVL7;
        signal Snot       : MVL7;
        signal RESETnot : MVL7;
        signal QQ         : MVL7;
        signal QQnot      : MVL7;
        signal NA1        : MVL7;

begin

        INV1: INVGATE
                generic map(Inverter_Delay, Inverter_Delay)
                port map(RESET,
                         RESETnot);             -- used to reset FF

        NAND1: NANDGATE
                generic map(2,                  -- 2 inputs
                            NAND_Delay,         -- rise inertial delay
                            NAND_Delay)         -- fall inertial delay
                port map(
                        input(1) => S,
                        input(2) => Rnot,
                        output   => Snot);


        NAND2: NANDGATE
                generic map(3,                  -- 3 inputs
                            NAND_Delay,         -- rise inertial delay
                            NAND_Delay)         -- fall inertial delay
                port map(
                        input(1) => Snot,
                        input(2) => CP,
                        input(3) => RESETnot,
                        output   => S);
```

```
NAND3: NANDGATE
        generic map(3,                  -- 3 inputs
                    NAND_Delay,         -- rise inertial delay
                    NAND_Delay)         -- fall inertial delay
        port map(
                input(1) => S,
                input(2) => NA1,
                input(3) => Rnot,
                output   => R);


NAND4: NANDGATE
        generic map(3,                  -- 3 inputs
                    NAND_Delay,         -- rise inertial delay
                    NAND_Delay)         -- fall inertial delay
        port map(
                input(1) => R,
                input(2) => D,
                input(3)  => RESETnot,
                output   => Rnot);


NAND5: NANDGATE
        generic map(2,                  -- 2 inputs
                    NAND_Delay,         -- rise inertial delay
                    NAND_Delay)         -- fall inertial delay
        port map(
                input(1) => S,
                input(2) => QQnot,
                output   => QQ);


NAND6: NANDGATE
        generic map(2,                  -- 2 inputs
                    NAND_Delay,         -- rise inertial delay
                    NAND_Delay)         -- fall inertial delay
        port map(
                input(1) => QQ,
                input(2) => R,
                output   => QQnot);


NAND7: NANDGATE
        generic map(2,                  -- 2 inputs
                    NAND_Delay,         -- rise inertial delay
                    NAND_Delay)         -- fall inertial delay
        port map(
                input(1) => CPnot,
                input(2) => RESETnot,
                output   => NA1);
```

```
    Q    <= QQ;                -- connect output port to output signal
    Qnot <= QQnot;             -- connect output port to output signal

end structure;
```

JK_FLIPFLOP

```
--------------------------------------------------------------------------
-- Date:    24 July 91
-- Version: 1.0
--
-- Filename: JK_flipflop.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the JK FF.
--
-- Associated files:
--       chip_pkg.vhd : This file contains constants, variables, etc. needed
--                      for this file.
--
-- History:
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


entity JK_FLIPFLOP is
        port(  J      : in MVL7;
               K      : in MVL7;
               RESET: in MVL7;
               CP     : in MVL7;
               Q      : inout MVL7;
               Qnot   : inout MVL7);
end JK_FLIPFLOP;


architecture structure of JK_FLIPFLOP is

--------------------------------------------------------------------------
-- This component is the AND gate.
--------------------------------------------------------------------------
        component ANDGATE                                  -- ZYCAD component
                generic (N: Positive;                      -- N input AND gate
                         tLH: Time;                        -- rise inertial delay
                         tHL: Time);                       -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);       -- N inputs
                         output: out MVL7);                -- one output
        end component;
```

```
-------------------------------------------------------------------------
-- This component is the NOR gate.
-------------------------------------------------------------------------
        component NORGATE                                   -- ZYCAD component
                generic (N: Positive;                       -- N input NOR gate
                         tLH: Time;                         -- rise inertial delay
                         tHL: Time);                        -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                         output: out MVL7);                 -- one output
        end component;

-------------------------------------------------------------------------
-- This component is the Inverter gate.
-------------------------------------------------------------------------
        component INVGATE                                   -- ZYCAD component
                generic (tLH: Time;                         -- rise inertial delay
                         tHL: Time);                        -- fall inertial delay
                port(input: in MVL7;                        -- N inputs
                         output: out MVL7);                 -- one output
        end component;

        signal Jout       : MVL7;
        signal Kout       : MVL7;
        signal RESETnot : MVL7;

begin
        INV1: INVGATE
                generic map(Inverter_Delay, Inverter_Delay)
                port map(RESET,
                         RESETnot);             -- used to reset FF

        AND1: ANDGATE
                generic map(3,                              -- 3 inputs
                            AND_Delay,                      -- rise inertial delay
                            AND_Delay)                      -- fall inertial delay
                port map(
                        input(1) => Q,
                        input(2) => K,
                        input(3) => CP,
                        output   => Kout);

        AND2: ANDGATE
                generic map(4,                              -- 4 inputs
                            AND_Delay,                      -- rise inertial delay
                            AND_Delay)                      -- fall inertial delay
                port map(
                        input(1) => Qnot,
                        input(2) => J,
                        input(3) => CP,
                        input(4) => RESETnot,
                        output   => Jout);
```

```
        NOR1: NORGATE
                generic map(3,                          -- 3 inputs
                        NOR_Delay,                      -- rise inertial delay
                        NOR_Delay)                      -- fall inertial delay
                port map(
                        input(1) => Kout,
                        input(2) => Qnot,
                        input(3) => RESET,
                        output   => Q);

        NOR2: NORGATE
                generic map(2,                          -- 2 inputs
                        NOR_Delay,                      -- rise inertial delay
                        NOR_Delay)                      -- fall inertial delay
                port map(
                        input(1) => Jout,
                        input(2) => Q,
                        output   => Qnot);

end structure;
```

MS_JKFF

```
--------------------------------------------------------------------------
-- Date:    30 July 91
-- Version: 1.0
--
-- Filename: ms_jkff.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the master-
--              slave JK FF.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc. needed
--                     for this file.
--
-- History:
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


entity MS_JKFF is
        port(   J       : in MVL7;
                K       : in MVL7;
                RESET: in MVL7;
                CP      : in MVL7;
                CPnot   : in MVL7;
                Q       : inout MVL7;
                Qnot    : inout MVL7);
end MS_JKFF;

architecture structure of MS_JKFF is

--------------------------------------------------------------------------
-- This component is the NAND gate.
--------------------------------------------------------------------------
        component NANDGATE                              -- ZYCAD component
                generic (N: Positive;                   -- N input NAND gate
                          tLH: Time;                    -- rise inertial delay
                          tHL: Time);                   -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);    -- N inputs
                          output: out MVL7);            -- one output
        end component;
```

```
-------------------------------------------------------------------------
-- This component is the Inverter gate.
-------------------------------------------------------------------------
        component INVGATE                       -- ZYCAD component
                generic (tLH: Time;             -- rise inertial delay
                         tHL: Time);            -- fall inertial delay
                port(input: in MVL7;            -- N inputs
                     output: out MVL7);         -- one output
        end component;


        signal Jout       : MVL7;
        signal Kout       : MVL7;
        signal Y          : MVL7;
        signal Ynot       : MVL7;
        signal Yout       : MVL7;
        signal Ynotout    : MVL7;
        signal RESETnot : MVL7;

begin

        INV1: INVGATE
                generic map(Inverter_Delay, -- rise inertial delay
                            Inverter_Delay)     -- fall inertial delay
                port map(
                        input  => RESET,
                        output => RESETnot);


        NAND1: NANDGATE
                generic map(4,                  -- 4 inputs
                            NAND_Delay,         -- rise inertial delay
                            NAND_Delay)         -- fall inertial delay
                port map(
                        input(1) => Qnot,
                        input(2) => J,
                        input(3) => CP,
                        input(4) => RESETnot,
                        output   => Jout);


        NAND2: NANDGATE
                generic map(3,                  -- 3 inputs
                            NAND_Delay,         -- rise inertial delay
                            NAND_Delay)         -- fall inertial delay
                port map(
                        input(1) => Q,
                        input(2) => K,
                        input(3) => CP,
                        output   => Kout);
```

```
NAND3: NANDGATE
        generic map(2,                          -- 2 inputs
                    NAND_Delay,                 -- rise inertial delay
                    NAND_Delay)                 -- fall inertial delay
        port map(
                input(1) => Jout,
                input(2) => Ynot,
                output   => Y);


NAND4: NANDGATE
        generic map(3,                          -- 3 inputs
                    NAND_Delay,                 -- rise inertial delay
                    NAND_Delay)                 -- fall inertial delay
        port map(
                input(1) => Y,
                input(2) => Kout,
                input(3) => RESETnot,
                output   => Ynot);


NAND5: NANDGATE
        generic map(2,                          -- 2 inputs
                    NAND_Delay,                 -- rise inertial delay
                    NAND_Delay)                 -- fall inertial delay
        port map(
                input(1) => Y,
                input(2) => CPnot,
                output   => Yout);


NAND6: NANDGATE
        generic map(2,                          -- 2 inputs
                    NAND_Delay,                 -- rise inertial delay
                    NAND_Delay)                 -- fall inertial delay
        port map(
                input(1) => Ynot,
                input(2) => CPnot,
                output   => Ynotout);


NAND7: NANDGATE
        generic map(2,                          -- 2 inputs
                    NAND_Delay,                 -- rise inertial delay
                    NAND_Delay)                 -- fall inertial delay
        port map(
                input(1) => Yout,
                input(2) => Qnot,
                output   => Q);
```

```
    NAND8: NANDGATE
        generic map(2,                      -- 2 inputs
                    NAND_Delay,             -- rise inertial delay
                    NAND_Delay)             -- fall inertial delay
        port map(
                input(1) => Ynotout,
                input(2) => Q,
                output   => Qnot);

end structure;
```

PREFETCH_COUNTER

```
--------------------------------------------------------------------------
-- Date:    5 September 1991
-- Version: 1.11
--
-- Filename: prefetch_counter.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the prefetch
--              counter. The binary counter counts upward and is compared
--              using XNOR gates, for equivalence, against the output of the
--              register that holds the number of lines to prefetch. When
--              they are equal, the port "Counting" becomes a '0'. This
--              component allows a line of memory to be prefetched on every
--              clock cycle.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc. needed
--                     for this file.
--
-- History: Version 1.0 (26 July 1991)
--          Version 1.1 (16 August 1991) - changed inout mode of Counting
--              port to out and added signal Countingout to feed back into
--              the component.
--          Version 1.11 (5 September 1991) - changed prefetch block to
--              subtract 2 instead of 1 from Prefetch_Block_Size. This is
--              because the first read on a prefetch is not counted in the
--              prefetch cycle.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;
use ZYCAD.BV_ARITHMETIC.all;


entity PREFETCH_COUNTER is
        port(   Start     : in MVL7;
                CP        : in MVL7;
                CPnot     : in MVL7;
                Reset     : in MVL7;
                Counting  : out MVL7);
end PREFETCH_COUNTER;

architecture structure of PREFETCH_COUNTER is
```

```
------------------------------------------------------------------------
-- This component is the binary counter.
------------------------------------------------------------------------
        component BINARY_COUNTER
                port(Count_Enable: in MVL7;                -- '1' to allow count
                     CP            : in MVL7;              -- clock
                     CPnot         : in MVL7;              -- clock complement
                     CLEAR         : in MVL7;              -- cleas to all '0's
                     Output        : inout MVL7_Vector(    -- binary count by 1
                                     Bits_in_Counter-1 downto 0));
        end component;


------------------------------------------------------------------------
-- This component is a D Flip-Flop register with reset.
------------------------------------------------------------------------
        component DFFREG                                   -- ZYCAD component
                generic (N: Positive;                      -- N input register
                         tLH: Time;                        -- rise inertial delay
                         tHL: Time);                       -- fall inertial delay
                port   (Data   : in MVL7_Vector;           -- data in
                         clock  : in  MVL7;                -- clock port
                         Reset  : in  MVL7;                -- reset port
                         Output: out MVL7_Vector);         -- data out
        end component;


------------------------------------------------------------------------
-- This component is the AND gate.
------------------------------------------------------------------------
        component ANDGATE                                  -- ZYCAD component
                generic (N: Positive;                      -- N input AND gate
                         tLH: Time;                        -- rise inertial delay
                         tHL: Time);                       -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);       -- N inputs
                         output: out MVL7);                -- one output
        end component;


------------------------------------------------------------------------
-- This component is the OR gate.
------------------------------------------------------------------------
        component ORGATE                                   -- ZYCAD component
                generic (N: Positive;                      -- N input AND gate
                         tLH: Time;                        -- rise inertial delay
                         tHL: Time);                       -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);       -- N inputs
                         output: out MVL7);                -- one output
        end component;
```

```
-------------------------------------------------------------------------
-- This component is the inverter gate.
-------------------------------------------------------------------------
        component INVGATE                       -- ZYCAD component
                generic (tLH: Time;             -- rise inertial delay
                         tHL: Time);            -- fall inertial delay
                port    (input: in MVL7;        -- N inputs
                         output: out MVL7);     -- one output
        end component;


-------------------------------------------------------------------------
-- This component is the exclusive NOR gate.
-------------------------------------------------------------------------
        component NXORGATE                              -- ZYCAD component
                generic (N: Positive;                   -- N input XNOR gate
                         tLH: Time;                     -- rise inertial delay
                         tHL: Time);                    -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);    -- N inputs
                         output: out MVL7);             -- one output
        end component;


-------------------------------------------------------------------------
-- This component is the RS FF.
-------------------------------------------------------------------------
        component RS_FLIPFLOP
                port(   R    : in MVL7;             -- R input
                        S    : in MVL7;             -- S input
                        CP   : in MVL7;             -- clock
                        Q    : inout MVL7;          -- output
                        Qnot: inout MVL7);          -- output complement
        end component;


        signal Prefetch_Block: MVL7_Vector(Bits_in_Counter-1 downto 0);
                -- this vector holds the integer converted to MVL7_Vector
                -- that describes the number of prefetches to perform
        signal The_Count    : MVL7_Vector(Bits_in_Counter-1 downto 0);
                -- this signal is the output of the counter
        signal Prefetch_Size : MVL7_Vector(Bits_in_Counter-1 downto 0);
                -- this is the output of the register that holds prefetch size
        signal XNOR_Out         : MVL7_Vector(Bits_in_Counter-1 downto 0);
                -- output of NXOR gates
        signal STOP             : MVL7;         -- input to R of RS FF
        signal RSclockin        : MVL7;         -- clock input to RS FF
        signal Clear_Counter : MVL7;            -- clr input to binary counter
        signal Countingnot   : MVL7;
        signal Countingout   : MVL7;
```

```
begin

-- component instantiation

        Counter: BINARY_COUNTER
                port map(Count_Enable => Countingout,
                         CP           => CP,
                         CPnot        => CPnot,
                         CLEAR        => Clear_Counter,
                         Output       => The_Count);


        Prefetch_Register: DFFREG          -- holds prefetch block size
                generic map(Bits_in_Counter,
                            DFF_Delay,
                            DFF_Delay)
                port map  (Data   => Prefetch_Block,
                           clock  => CP,
                           Reset  => Reset,
                           Output => Prefetch_Size);

-------------------------------------------------------------------------
-- The following code generates the required number of XNOR gates to perform
--  the equivalence operation between counter and Prefetch_Register.
-------------------------------------------------------------------------
        X1:
        for I in Bits_in_Counter-1 downto 0 generate
                Equivalent: NXORGATE
                        generic map(2,                      -- 2 inputs
                                    XNOR_Delay,             -- rise inertial delay
                                    XNOR_Delay)             -- fall inertial delay
                        port map  (input(1) => Prefetch_Size(I),
                                   input(2) => The_Count(I),
                                   output   => XNOR_Out(I));
        end generate;

        AND1: ANDGATE
                generic map(Bits_in_Counter,        -- # inputs
                            AND_Delay,              -- rise inertial delay
                            AND_Delay)              -- fall inertial delay
                port map(input  => XNOR_Out,
                         output => Stop);

        OR1: ORGATE
                generic map(3,                      -- 3 inputs
                            OR_Delay,           -- rise inertial delay
                            OR_Delay)           -- fall inertial delay
                port map(input(1) => Start,
                         input(2) => Countingout,
                         input(3) => Reset,
                         output   => RSclockin);
```

```
OR2: ORGATE
        generic map(2,                  -- 2 inputs
                    OR_Delay,           -- rise inertial delay
                    OR_Delay)           -- fall inertial delay
        port map(input(1) => Reset,
                 input(2) => Countingnot,
                 output   => Clear_Counter);


INV1: INVGATE
        generic map(Inverter_Delay,         -- rise inertial delay
                    Inverter_Delay)         -- fall inertial delay
        port map  (input  => Countingout,
                   output  => Countingnot);


RSFF: RS_flipflop
        port map(R   => Stop,
                 S    => START,
                 CP   => RSclockin,
                 Q    => Countingout,
                 Qnot => OPEN);

-------------------------------------------------------------------------
-- This converts an integer to bit_vector then bit_vector to MVL7_Vector.
--  The '-1' accounts for the count starting at zero instead of one.
-------------------------------------------------------------------------
Prefetch_Block <= BVtoMVL7V(ItoBV(Prefetch_Block_Size-1))
                                    (Bits_in_Counter-1 downto 0);

Counting     <= Countingout;                -- connects signal to output port


end structure;
```

RS_FLIPFLOP

```
-------------------------------------------------------------------------
-- Date:    23 July 91
-- Version: 1.0
--
-- Filename: RS_flipflop.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the RS FF.
--
-- Associated files:
--       chip_pkg.vhd : This file contains constants, variables, etc. needed
--                      for this file.
--
-- History:
--
-- Author: Curtis M. Winstead
-------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


entity RS_FLIPFLOP is
        port(   R    : in MVL7;
                S    : in MVL7;
                CP   : in MVL7;
                Q    : inout MVL7 := '0';
                Qnot: inout MVL7 := '1');
end RS_FLIPFLOP;


architecture structure of RS_FLIPFLOP is

-------------------------------------------------------------------------
-- This component is the AND gate.
-------------------------------------------------------------------------
        component ANDGATE                                   -- ZYCAD component
                generic (N: Positive;                       -- N input AND gate
                        tLH: Time;                          -- rise inertial delay
                        tHL: Time);                         -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                        output: out MVL7);                  -- one output
        end component;
```

```
--------------------------------------------------------------------------
-- This component is the NOR gate.
--------------------------------------------------------------------------
        component NORGATE                                   -- ZYCAD component
                generic (N: Positive;                       -- N input NOR gate
                         tLH: Time;                         -- rise inertial delay
                         tHL: Time);                        -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                     output: out MVL7);                     -- one output
        end component;

        signal Rout: MVL7;
        signal Sout: MVL7;
begin
        AND1: ANDGATE
                generic map(2,                              -- 2 inputs
                            AND_Delay,                      -- rise inertial delay
                            AND_Delay)                      -- fall inertial delay
                port map(
                        input(1) => R,
                        input(2) => CP,
                        output   => Rout);

        AND2: ANDGATE
                generic map(2,                              -- 2 inputs
                            AND_Delay,                      -- rise inertial delay
                            AND_Delay)                      -- fall inertial delay
                port map(
                        input(1) => S,
                        input(2) => CP,
                        output   => Sout);

        NOR1: NORGATE
                generic map(2,                              -- 2 inputs
                            NOR_Delay,                      -- rise inertial delay
                            NOR_Delay)                      -- fall inertial delay
                port map(
                        input(1) => Rout,
                        input(2) => Qnot,
                        output   => Q);

        NOR2: NORGATE
                generic map(2,                              -- 2 inputs
                            NOR_Delay,                      -- rise inertial delay
                            NOR_Delay)                      -- fall inertial delay
                port map(
                        input(1) => Sout,
                        input(2) => Q,
                        output   => Qnot);

end structure;
```

SHIFT_REGISTER

```
--------------------------------------------------------------------------
-- Date:    4 September 1991
-- Version:  1.1
--
-- Filename:  shift_register.vhd
-- System:    ZYCAD, VLSI net
-- Language:  VHDL
--
-- Description:  This file contains the entity and structure of a shift
--       register.  This register is generic, i.e. you can make it any size you
--       want.  The shift register uses a D flip-flop and a 2x1 MUX.  It is a
--       circular shift register run by a clock.
--
-- Associated files:
--       chip_pkg.vhd :  This file contains constants,
--                       variables, etc. needed for this file.
--
-- History:  Version 1.0 (5 July 1991)
--           Version 1.1 (4 September 1991) - changed the Sel0 to shift on a
--               '1' and load on a '0'.
--
-- Author:  Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


--------------------------------------------------------------------------
-- The entity of the shift register.
--------------------------------------------------------------------------
entity SHIFT_REGISTER is
        generic(Size: Positive);                    -- make it any size you want
        port  (In_Vector : in MVL7_Vector;          -- input vector
               Sel0      : in MVL7;                 -- '1' is a load, '0' shift
               Clockin   : in MVL7;                 -- clock port
               Clear     : in MVL7;                 -- clear the registers
               SR_Output: inout MVL7_Vector);       -- ouput vector, is of mode in
                                                    -- to feed back into MUX
                                                    -- allowing for circular shift
end SHIFT_REGISTER;


--------------------------------------------------------------------------
-- Architecture of shift register.
--------------------------------------------------------------------------
architecture structure of SHIFT_REGISTER is
```

```
        signal MUX_Out : MVL7_Vector(Size-1 downto 0);

-------------------------------------------------------------------------
-- This component is used to determine if the shift register is doing a shift
-- or a parallel load. If Sel is a '0' then a shift is performed. If Sel is
-- a '1' then a parallel load is performed.
-------------------------------------------------------------------------
        component MUX2x1                    -- ZYCAD component
                generic (tLH: Time;         -- rise inertial delay
                         tHL: Time);        -- fall inertial delay
                port   (In0    : in MVL7;   -- data input 0
                        In1    : in MVL7;   -- data input 1
                        Sel    : in MVL7;   -- select input (0 => In0)
                        Output: out MVL7); -- output
        end component;

-------------------------------------------------------------------------
-- This component is a D Flip-Flop register with reset.
-------------------------------------------------------------------------
        component DFFREG                            -- ZYCAD component
                generic (N : Positive;              -- N input register
                         tLH: Time;                 -- rise inertial delay
                         tHL: Time);                -- fall inertial delay
                port   (Data   : in MVL7_Vector;    -- input vector
                        clock  : in MVL7;           -- clock port
                        Reset  : in MVL7;           -- reset
                        Output: out MVL7_Vector);  -- output vector
        end component;

begin
-------------------------------------------------------------------------
-- The following code generates a set of MUXs of size Size. Size can be set
-- to any value.
-------------------------------------------------------------------------
        M1:
        for I in Size-1 downto 0 generate
                ---------------------------------------------------------
                -- This conditional generate creates all MUXs of the desired
                -- size except for the 0th MUX. The reason is that the
                -- 0th MUX needs to be fed the output of the Size-1 D flip-
                -- flop register (SR_Output).
                ---------------------------------------------------------
                M2:
                if I /= 0 generate
                        MUX_1toSize_minus_1: MUX2x1
                                generic map (MUX_Delay, MUX_Delay)
                                port map (In0    => In_Vector(I),
                                          In1    => SR_Output(I-1),
                                          Sel    => Sel0,
                                          Output => MUX_Out(I));
                end generate;
```

```
            ------------------------------------------------------------
            -- This was kept inside the generate function for associative
            -- purposes. It is the 0th MUX and its In0 is connected to
            -- the Size-1 output of the D flip-flop register (SR_Output).
            ------------------------------------------------------------
            M3:
            if I = 0 generate
                  MUX_0: MUX2x1
                        generic map (MUX_Delay, MUX_Delay)
                        port map (In0    => In_Vector(I),
                                  In1    => SR_Output(Size-1),
                                  Sel    => Sel0,
                                  Output => MUX_Out(I));
            end generate;

      end generate;

------------------------------------------------------------------------
-- This register holds the values of the shift register.
------------------------------------------------------------------------
      Reg: DFFREG
            generic map(
                  Size,
                  DFF_Delay,
                  DFF_Delay)
            port map(
                  Data   => MUX_Out,
                  clock  => Clockin,
                  Reset  => Clear,
                  Output => SR_Output);

end structure;
```

TOS_SHIFTER

```
-------------------------------------------------------------------------
-- Date:    20 Aug 1991
-- Version:  1.2
--
-- Filename:  TOS_shifter.vhd
-- System:    ZYCAD, VLSI net
-- Language:  VHDL
--
-- Description:  This file contains the entity and structure of a shift
--       register.  This register is generic, i.e. you can make it any size you
--       want.  The shift register uses edge triggered D flip-flops.  It is a
--       circular shift register run by a clock.  One unique feature which may
--       be a problem is that it takes 2 ns to stabilize; an inadvertent write
--       is possible.  It is used to hold the top of stack pointer for the
--       word select register used to write into the CAM.
--
-- Associated files:
--       chip_pkg.vhd :  This file contains constants,
--                       variables, etc. needed for this file.
--
-- History:  Version 1.0 (1 Aug 91)
--           Version 1.1 (5 Aug 91) - modified component to clear outputs and
--               load first D FF with a '1' on initial clear.
--           Version 1.2 (20 Aug 91) - modified to handle additional CPnot
--               port of edge_triggered_DFF used to help reset the FF.
--
-- Author:  Curtis M. Winstead
-------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


-------------------------------------------------------------------------
-- The entity of the shift register.
-------------------------------------------------------------------------
entity TOS_SHIFTER is
        generic(Size: Positive);                    -- make it any size you want
        port  (CLEAR : in MVL7;                     -- clears FFs
               CP    : in MVL7;                     -- clock
               CPnot : in MVL7;                     -- clock complement
               Output: inout MVL7_Vector);          -- ouput vector, is of mode in
                                                    -- to feed back into MUX
                                                    -- allowing for circular shift
end TOS_SHIFTER;
```

```
-------------------------------------------------------------------------
-- Architecture of TOS shift register.
-------------------------------------------------------------------------
architecture structure of TOS_SHIFTER is


-- Internal signals of TOS shift register.
        signal Din0   : MVL7;
        signal CPin0  : MVL7;
        signal Tied_Low: MVL7 := '0';


-------------------------------------------------------------------------
-- This component is the edge triggered DFF.
-------------------------------------------------------------------------
        component EDGE_TRIGGERED_DFF
                port(D     : in MVL7;
                     RESET : in MVL7;
                     CP    : in MVL7;
                     CPnot : in MVL7;
                     Q     : inout MVL7;
                     Qnot  : inout MVL7);
        end component;


-------------------------------------------------------------------------
-- This component is the OR gate.
-------------------------------------------------------------------------
        component ORGATE                                    -- ZYCAD component
                generic (N: Positive;                       -- N input OR gate
                         tLH: Time;                         -- rise inertial delay
                         tHL: Time);                        -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                     output: out MVL7);                     -- one output
        end component;


begin

-------------------------------------------------------------------------
-- The following code generates the desired number of edge triggered D FFs
--  in the TOS pointer.
-------------------------------------------------------------------------
        D1:
        for I in Size-1 downto 0 generate
```

```
        ------------------------------------------------------------
        -- The following code isolates the very first, or bottom, of
        -- the stack and allows it to be loaded with a '1' to start
        -- the pointer.
        ------------------------------------------------------------
        D2:
        if I = 0 generate
                DFF0: EDGE_TRIGGERED_DFF
                        port map(D      => Din0,
                                 RESET => Tied_Low,
                                 CP     => CPin0,
                                 CPnot  => CPnot,
                                 Q      => Output(0),
                                 Qnot   => OPEN);
        end generate;


        ------------------------------------------------------------
        -- The remaining code is the rest of the edge triggered D FFs.
        -- They hold the stack pointer after receiving it from the
        -- previous FF.
        ------------------------------------------------------------
        D3:
        if I/= 0 generate
                DFFnot0: EDGE_TRIGGERED_DFF
                        port map(D      => Output(I-1),
                                 RESET => CLEAR,
                                 CP     => CP,
                                 CPnot  => CPnot,
                                 Q      => Output(I),
                                 Qnot   => OPEN);
        end generate;

end generate;


------------------------------------------------------------------
-- This OR gate loads the bottom of the stack with either the CLEAR
-- signal or the value of the top of the stack (D FF).
------------------------------------------------------------------
OR1: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => CLEAR,
                 input(2) => Output(Size-1),
                 output   => Din0);
```

```
        ------------------------------------------------------------------
        -- This OR gate feeds the clock input of the bottom of the stack.
        -- On the initial CLEAR, a '1' is loaded; the rest of the time the
        -- clock pulse clocks the D FF.
        ------------------------------------------------------------------
        OR2: ORGATE
                generic map(2,                          -- 2 inputs
                            OR_Delay,                   -- rise inertial delay
                            OR_Delay)                   -- fall inertial delay
                port map(input(1) => CLEAR,
                         input(2) => CP,
                         output   => CPin0);

end structure;
```

Appendix C: The VHDL Code for the Major Components

This appendix contains the major components that make up the controller section of the MCC. They are listed in alphabetical order and begin on separate pages.

FUNCTION_CHANGE_DETECTOR

```
-------------------------------------------------------------------------
-- Date:    12 September 1991
-- Version: 1.01
--
-- Filename: function_change_detector.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the
--              function_change_detector component. Its purpose is to
--              detect the beginning of a read or write cycle.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc. needed
--                     for this file.
--
-- History: Version 1.0 (22 August 1991)
--          Version 1.01 (12 September 1991) - changed the name from clear_
--              word_select_register to FUNCTION_CHANGE_DETECTOR.
--
-- Author: Curtis M. Winstead
-------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


entity FUNCTION_CHANGE_DETECTOR is
        port(   Read            : in MVL7;
                Write           : in MVL7;
                Function_Change: out MVL7);
end FUNCTION_CHANGE_DETECTOR;


architecture structure of FUNCTION_CHANGE_DETECTOR is

        signal Rd_out: MVL7;
        signal Wt_out: MVL7;
```

```
-------------------------------------------------------------------------
-- This component is the OR gate.
-------------------------------------------------------------------------
        component ORGATE                                    -- ZYCAD component
                generic (N: Positive;                       -- N input OR gate
                        tLH: Time;                          -- rise inertial delay
                        tHL: Time);                         -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                        output: out MVL7);                  -- one output
        end component;


-------------------------------------------------------------------------
-- This component is the change detector.
-------------------------------------------------------------------------
        component CHANGE_DETECTOR
                generic(Delay: Time);
                port(Input : in MVL7;
                   Output: out MVL7);
        end component;


begin

-- component instantiation

        Read_cd: CHANGE_DETECTOR
                generic map(Change_Detector_Delay)
                port map(Input  => Read,
                        Output => Rd_out);



        Write_cd: CHANGE_DETECTOR
                generic map(Change_Detector_Delay)
                port map(Input  => Write,
                        Output => Wt_out);



        O1: ORGATE
                generic map(2,                              -- 2 inputs
                          OR_Delay,                         -- rise inertial delay
                          OR_Delay)                         -- fall inertial delay
                port map(input(1) => Rd_out,
                        input(2) => Wt_out,
                        output   => Function_Change);


end structure;
```

OPERATION_STATUS

```
--------------------------------------------------------------------------
-- Date:    4 September 1991
-- Version: 1.1
--
-- Filename: operation_status.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the
--              operation_status component of the CAM chip controller.
--              Its purpose is to output the status of the chip: 1) data
--              available, 2) read miss, 3) write hit, and 4) write miss.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc.
--                     needed for this file.
--
-- History: Version 1.0 (13 August 1991)
--          Version 1.1 (4 September 1991) - added port Counting.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;

entity OPERATION_STATUS is
        port(
                Data                : in MVL7_Vector;
                Prefetching         : in MVL7;
                Read                : in MVL7;
                Write               : in MVL7;
                Resolved_Tags       : in MVL7;
                Search_Complete     : in MVL7;
                Counting            : in MVL7;
                Data_Out_Available  : out MVL7;   -- read hit
                Read_Miss           : out MVL7;   -- read miss
                Write_Miss          : out MVL7;   -- write miss
                Write_Hit           : out MVL7);  -- write hit
end OPERATION_STATUS;


architecture structure of OPERATION_STATUS is

        signal Resolved_TagsNot : MVL7;
        signal Data_Change      : Vector_Word_length;
        signal Data_Out_Change  : Wired_Or_Type;
```

```
-------------------------------------------------------------------------
-- This component detects a change in the signal.
-------------------------------------------------------------------------
        component CHANGE_DETECTOR
                generic(Delay: Time);
                port(Input : in MVL7;
                     Output: out MVL7);
        end component;

-------------------------------------------------------------------------
-- This component is the AND gate.
-------------------------------------------------------------------------
     component ANDGATE                                -- ZYCAD component
          generic (N: Positive;                       -- N input AND gate
                tLH: Time;                            -- rise inertial delay
                tHL: Time);                           -- fall inertial delay
          port(input: in MVL7_VECTOR (1 to N);        -- N inputs
               output: out MVL7);                     -- one output
     end component;

-------------------------------------------------------------------------
-- This component is the Inverter gate.
-------------------------------------------------------------------------
     component INVGATE              -- ZYCAD component
          generic (tLH: Time;       -- rise inertial delay
                tHL: Time);         -- fall inertial delay
          port(input: in MVL7;      -- N inputs
               output: out MVL7);   -- one output
     end component;

begin
        -- component instantiations

        -------------------------------------------------------------------
        -- The following code generates the change detectors to detect if a
        --  change has occurred on the input lines.
        -------------------------------------------------------------------
        CD:
        for I in Word_length-1 downto 0 generate
                change_detectors: CHANGE_DETECTOR
                        generic map(Change_Detector_Delay)
                        port map(Input  => Data(I),
                                 Output => Data_Change(I));

        -------------------------------------------------------------------
        -- This code resolves, by Wired_Or, the signals of Data_Change into
        --  one signal.
        -------------------------------------------------------------------
        Data_Out_Change <= Data_Change(I);

        end generate;
```

```
    INV1: INVGATE
            generic map(Inverter_Delay, Inverter_Delay)
            port map(input  => Resolved_Tags,
                     output => Resolved_TagsNot);

    AND1: ANDGATE
            generic map(2, AND_Delay, AND_Delay)
            port map(input(1) => Data_Out_Change,
                     input(2) => Prefetching,
                     output   => Data_Out_Available);

    AND2: ANDGATE
            generic map(3, AND_Delay, AND_Delay)
            port map(input(1) => Read,
                     input(2) => Resolved_TagsNot,
                     input(3) => Search_Complete,
                     output   => Read_Miss);

    AND3: ANDGATE
            generic map(3, AND_Delay, AND_Delay)
            port map(input(1) => Resolved_TagsNot,
                     input(2) => Search_Complete,
                     input(3) => Write,
                     output   => Write_Miss);

    AND4: ANDGATE
            generic map(3, AND_Delay, AND_Delay)
            port map(input(1) => Write,
                     input(2) => Resolved_Tags,
                     input(3) => Search_Complete,
                     output   => Write_Hit);


end structure;
```

PREFETCH_STATUS

```
--------------------------------------------------------------------------
-- Date:    16 August 1991
-- Version: 1.0
--
-- Filename: prefetch_status.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the PREFETCH_
--              STATUS component. Its purpose is to output a '1' when the
--              prefetch counter is counting the number of clock pulses
--              equal to the number of lines to prefetch from the CAM.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc. needed
--                     for this file.
-- History:
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;
use ZYCAD.BV_ARITHMETIC.all;

entity PREFETCH_STATUS is
        port(   CP                      : in MVL7;
                CPnot                   : in MVL7;
                Read                    : in MVL7;
                Resolved_Tags           : in MVL7;
                Search_Complete         : in MVL7;
                Reset                   : in MVL7;  -- must be asserted at least 19 ns
                Counting                : out MVL7);
end PREFETCH_STATUS;

architecture structure of PREFETCH_STATUS is

        signal Start: MVL7;

--------------------------------------------------------------------------
-- This component is the AND gate.
--------------------------------------------------------------------------
        component ANDGATE                                       -- ZYCAD component
                generic (N: Positive;                           -- N input AND gate
                        tLH: Time;                              -- rise inertial delay
                        tHL: Time);                             -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);            -- N inputs
                        output: out MVL7);                      -- one output
        end component;
```

```
-------------------------------------------------------------------------
-- This component is the PREFETCH_COUNTER that when started produces an
--  output of '1' until the clock pulses prefetch_size times.
-------------------------------------------------------------------------
        component PREFETCH_COUNTER
                port(   Start     : in MVL7;
                        CP        : in MVL7;
                        CPnot     : in MVL7;
                        Reset     : in MVL7;
                        Counting: out MVL7);
        end component;


begin

-- component instantiation

        Count: PREFETCH_COUNTER
                port map(Start    => Start,
                         CP       => CP,
                         CPnot    => CPnot,
                         Reset    => Reset,
                         Counting => Counting);



        A1: ANDGATE
                generic map(3,                          -- 3 inputs
                            AND_Delay,                  -- rise inertial delay
                            AND_Delay)                  -- fall inertial delay
                port map(input(1) => Read,
                         input(2) => Resolved_Tags,
                         input(3) => Search_Complete,
                         output   => Start);


end structure;
```

SEARCH_STATUS

```
--------------------------------------------------------------------------
-- Date:    5 September 1991
-- Version: 1.3
--
-- Filename: search_status.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the
--              SEARCH_STATUS component of the CAM chip controller.  Its
--              purpose is to determine when the search cycle is complete.
--
-- Associated files:
--      chip_pkg.vhd :  This file contains constants, variables, etc.
--                      needed for this file.
--
-- History:  Version 1.0 (14 August 1991)
--           Version 1.1 (29 August 1991) - deleted the Search_Delay component.
--           Version 1.2 (3 September 1991) - added a DFF to hold the state
--               that the search has been completed.
--           Version 1.3 (5 September 1991) - added the port Counting to clear
--               the DFF flip-flop when the chip is prefetching.  This prevents
--               the prefetch counter from resetting and starting over again.
--
-- Author:  Curtis M. Winstead
--------------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;

entity SEARCH_STATUS is
        port(
                Data            : in MVL7_Vector;
                Read            : in MVL7;
                Write           : in MVL7;
                Function_Change : in MVL7;
                Reset_DFF       : in MVL7;
                Counting        : in MVL7;
                Search_Complete : out MVL7);
end SEARCH_STATUS;


architecture structure of SEARCH_STATUS is

        signal Address_Change  : Vector_Address_length;
        signal Or_Out1         : MVL7;
        signal Or_Out2         : MVL7;
        signal Or_Out2not      : MVL7;
```

```
        signal Search_Done      : MVL7;
        signal NOR_Out          : MVL7;
        signal AND_Out          : MVL7;
        signal Res_DFF          : MVL7;

        signal Resolved_Signal_Address_Change: Wired_Or_Type;


-------------------------------------------------------------------------
-- This component detects a change in the signal.
-------------------------------------------------------------------------
        component CHANGE_DETECTOR
                generic(Delay: Time);
                port(Input : in MVL7;
                   Output: out MVL7);
        end component;


-------------------------------------------------------------------------
-- This component is the edge triggered DFF.
-------------------------------------------------------------------------
        component EDGE_TRIGGERED_DFF
                port(   D       : in MVL7;
                        RESET: in MVL7;
                        CP      : in MVL7;
                        CPnot   : in MVL7;
                        Q       : out MVL7;
                        Qnot    : out MVL7);
        end component;

-------------------------------------------------------------------------
-- This component is the AND gate.
-------------------------------------------------------------------------
      component ANDGATE                                 -- ZYCAD component
          generic (N: Positive;                         -- N input AND gate
                tLH: Time;                              -- rise inertial delay
                tHL: Time);                             -- fall inertial delay
          port(input: in MVL7_VECTOR (1 to N);          -- N inputs
                output: out MVL7);                      -- one output
      end component;

-------------------------------------------------------------------------
-- This component is the OR gate.
-------------------------------------------------------------------------
      component ORGATE                                  -- ZYCAD component
          generic (N: Positive;                         -- N input OR gate
                tLH: Time;                              -- rise inertial delay
                tHL: Time);                             -- fall inertial delay
          port(input: in MVL7_VECTOR (1 to N);          -- N inputs
                output: out MVL7);                      -- one output
      end component;
```

```
-------------------------------------------------------------------------
-- This component is the NOR gate.
-------------------------------------------------------------------------
    component NORGATE                              -- ZYCAD component
        generic (N: Positive;                      -- N input NOR gate
                 tLH: Time;                        -- rise inertial delay
                 tHL: Time);                       -- fall inertial delay
        port(input: in MVL7_VECTOR (1 to N);       -- N inputs
             output: out MVL7);                    -- one output
    end component;


-------------------------------------------------------------------------
-- This component is the Inverter gate.
-------------------------------------------------------------------------
    component INVGATE                              -- ZYCAD component
        generic (tLH: Time;                        -- rise inertial delay
                 tHL: Time);                       -- fall inertial delay
        port(        input: in MVL7;               -- input
             output: out MVL7);                    -- output
    end component;

begin
        -- component instantiations

        -------------------------------------------------------------------
        -- The following code generates the change detectors to detect if a
        --  change has occurred on the input lines.
        -------------------------------------------------------------------
        CD:
        for I in Word_length-1 downto Data_length generate
                search_stat_change_detector: CHANGE_DETECTOR
                        generic map(Change_Detector_Delay)
                        port map(Input  => Data(I),
                                 Output => Address_Change(I));

        -------------------------------------------------------------------
        -- This code resolves, by Wired_Or, the signals of Address_Change
        --  into one signal.
        -------------------------------------------------------------------
        Resolved_Signal_Address_Change <= Address_Change(I);

        end generate;

        DFF: EDGE_TRIGGERED_DFF
                port map(D      => Search_Done,
                         RESET  => Res_DFF,
                         CP     => Or_Out2,
                         CPnot  => Or_Out2not,
                         Q      => Search_Complete,
                         Qnot   => OPEN);
```

```
        OR1: ORGATE
                generic map(2, OR_Delay, OR_Delay)
                port map(input(1) => Read,
                         input(2) => Write,
                         output   => Or_Out1);

        OR2: ORGATE
                generic map(2, OR_Delay, OR_Delay)
                port map(input(1) => Search_Done,
                         input(2)  => AND_Out,
                         output   => Or_Out2);

        OR3: ORGATE
                generic map(3, OR_Delay, OR_Delay)
                port map(input(1) => Reset_DFF,
                         input(2) => AND_Out,
                         input(3) => Counting,
                         output   => Res_DFF);

        NOR1: NORGATE
                generic map(2, NOR_Delay, NOR_Delay)
                port map(input(1) => Read,
                         input(2) => Write,
                         output   => NOR_Out);

        AND1: ANDGATE
                generic map(2, AND_Delay, AND_Delay)
                port map(input(1) => Resolved_Signal_Address_Change,
                         input(2) => Or_Out1,
                         output   => Search_Done);

        AND2: ANDGATE
                generic map(2, AND_Delay, AND_Delay)
                port map(input(1) => NOR_Out,
                         input(2) => Function_Change,
                         output   => AND_Out);

        INV1: INVGATE
                generic map(Inverter_Delay, Inverter_Delay)
                port map(input  => Or_Out2,
                        output => Or_Out2not);

end structure;
```

SELECT_WORD_SELECT

```
-------------------------------------------------------------------------
-- Date:    12 September 1991
-- Version: 1.1
--
-- Filename: select_word_select.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the SELECT_
--              WORD_SELECT component. Its purpose is to select the
--              correct word select register. The output is fed into the
--              mux that selects the correct one.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc. needed
--                     for this file.
--
-- History: Version 1.0 (22 August 1991)
--          Version 1.1 (30 August 1991) - added port Data_Avail_MEM to
--              determine if data are available from Main Memory on a Read
--              Miss.
--          Version 1.11 (12 September 1991) - changed port name from
--              Address_Change to Function_Change.
--
-- Author: Curtis M. Winstead
-------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;


entity SELECT_WORD_SELECT is
        port(   Read                    : in MVL7;
                Resolved_Tags           : in MVL7;
                Search_Complete         : in MVL7;
                Function_Change         : in MVL7;
                Master_Reset            : in MVL7;
                Data_Avail_MEM          : in MVL7;
                WSR_Select              : out MVL7);
end SELECT_WORD_SELECT;

architecture structure of SELECT_WORD_SELECT is

        signal RT_not: MVL7;
        signal D_in  : MVL7;
        signal RST   : MVL7;
        signal Or_out : MVL7;
        signal OrNot  : MVL7;
```

```
-------------------------------------------------------------------------
-- This component is the AND gate.
-------------------------------------------------------------------------
        component ANDGATE                                   -- ZYCAD component
                generic (N: Positive;                       -- N input AND gate
                         tLH: Time;                         -- rise inertial delay
                         tHL: Time);                        -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                     output: out MVL7);                     -- one output
        end component;


-------------------------------------------------------------------------
-- This component is the OR gate.
-------------------------------------------------------------------------
        component ORGATE                                    -- ZYCAD component
                generic (N: Positive;                       -- N input OR gate
                         tLH: Time;                         -- rise inertial delay
                         tHL: Time);                        -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                     output: out MVL7);                     -- one output
        end component;


-------------------------------------------------------------------------
-- This component is the Inverter gate.
-------------------------------------------------------------------------
        component INVGATE                                   -- ZYCAD component
                generic (tLH: Time;                         -- rise inertial delay
                         tHL: Time);                        -- fall inertial delay
                port    (input: in MVL7;                    -- input
                         output: out MVL7);                 -- output
        end component;


-------------------------------------------------------------------------
-- This is the edge_triggered_DFF component.
-------------------------------------------------------------------------
        component EDGE_TRIGGERED_DFF
                port(D     : in MVL7;
                     RESET: in MVL7;
                     CP    : in MVL7;
                     CPnot : in MVL7;
                     Q     : out MVL7;
                     Qnot  : out MVL7);
        end component;


begin

-- component instantiation
```

```
INV1: INVGATE
        generic map(Inverter_Delay, Inverter_Delay)
        port map(input  => Resolved_Tags,
                 output => RT_not);

INV2: INVGATE
        generic map(Inverter_Delay, Inverter_Delay)
        port map(input  => Or_out,
                 output => OrNot);

AND1: ANDGATE
        generic map(4,                          -- 4 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => Read,
                 input(2) => RT_not,
                 input(3) => Search_Complete,
                 input(4) => Data_Avail_MEM,
                 output   => D_in);

OR1: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => Function_Change,
                 input(2) => Master_Reset,
                 output   => RST);

OR2: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => D_in,
                 input(2) => RST,
                 output   => Or_out);

DFF: EDGE_TRIGGERED_DFF
        port map(D      => D_in,
                 RESET => RST,
                 CP     => Or_out,
                 CPnot  => OrNot,
                 Q      => WSR_Select,
                 Qnot   => OPEN);


end structure;
```

WORD_SELECT

```
-------------------------------------------------------------------------
-- Date:    11 September 1991
-- Version: 2.0
--
-- Filename: word_select.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the
--              WORD_SELECT component of the CAM chip. It
--              contains the shift registers that act as the word select
--              inputs to the CAM array.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc.
--                     needed for this file.
--
-- History: Version 1.0 (12 August 1991)
--          Version 1.1 (20 August 1991) - added port TOS_CPnot to
--              correspond to addition of port to edge-triggered DFF.
--          Version 1.2 (29 August 1991) - added an OR gate to clear the
--              DFF register of the Word_Selector with its output. Its
--              inputs are the Master_Reset and the Function_Change signal.
--              Had to change ports as a result.
--          Version 1.3 (4 September 1991) - added a DFF to hold the Sel0
--              line high during a read hit so the Word_Selector will shift
--              on the prefetch. Also added a CHANGE_DETECTOR to clock the
--              signal in.
--          Version 1.4 (5 September 1991) - added AND gate to feed the Clear
--              port of the Word_Selector. Now it can only be cleared when
--              not prefetching.
--          Version 1.5 (10 September 1991) - added a NOR gate to output to
--              the new Bit_Select port used to connect into the Bit_Select_
--              Bus.
--          Version 2.0 (11 September 1991) - This was a major change due
--              to adding the component controller to the cam_chip. Timing
--              was off and as a consequence data was being written into the
--              CAM array when it shouldn't have.
--
-- Author: Curtis M. Winstead
-------------------------------------------------------------------------

library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;
```

```
entity WORD_SELECT is
        port(
                Clear_TOS           : in MVL7;
                TOS_CP              : in MVL7;
                TOS_CPnot           : in MVL7;
                Word_Selector_CP    : in MVL7;
                WSR_Select          : in MVL7;
                Tags_In             : in MVL7_Vector;
                Clear1              : in MVL7;
                Clear2              : in MVL7;
                SR_Select           : in MVL7;
                Shifting            : out MVL7;
                Bit_Select          : out MVL7;
                Select_Word         : out MVL7_Vector);
end WORD_SELECT;

architecture structure of WORD_SELECT is
        signal TOS_Output       : Vector_Depth;
        signal SR_Output        : Vector_Depth;
        signal Or1_out          : MVL7;
        signal OR3_out          : MVL7;
        signal OR4_out          : MVL7;
        signal OR4_outnot       : MVL7;
        signal AND1_out         : MVL7;
        signal AND2_out         : MVL7;
        signal AND3_out         : MVL7;
        signal AND4_out         : MVL7;
        signal NOR1_out         : MVL7;
        signal CD2_out          : MVL7;
        signal CD3_out          : MVL7;
        signal CD4_out          : MVL7;
        signal DFF2_out         : MVL7;
        signal Word_Sel_CP      : MVL7;
        signal Selector         : MVL7;
        signal D_CP             : MVL7;
        signal D_CPnot          : MVL7;
        signal Selector_not     : MVL7;
        signal Select_Word_Signal : Vector_Depth;
        signal Resolved_Select_Word: Wired_Or_Type;
-------------------------------------------------------------------------------
-- This is the shift register that shifts up only.  It is used to keep track
--  of the TOS for writing into the chip.
-------------------------------------------------------------------------------
        component TOS_SHIFTER
                generic(Size: Positive);
                port(
                        CLEAR : in MVL7;
                        CP    : in MVL7;
                        CPnot : in MVL7;
                        Output:  inout MVL7_Vector);
        end component;
```

```
-------------------------------------------------------------------------
-- This component is a shift register with load capabilities.  It is used to
--  store the location of a word on a search hit.
-------------------------------------------------------------------------
        component SHIFT_REGISTER
                generic(Size: Positive);
                port(
                        In_Vector  : in MVL7_Vector;
                        Sel0       : in MVL7;
                        Clockin    : in MVL7;
                        Clear      : in MVL7;
                        SR_Output: inout MVL7_Vector);
        end component;


-------------------------------------------------------------------------
-- This is the ZYCAD component 2x1 MUX.
-------------------------------------------------------------------------
        component MUX2x1
                generic(tLH: Time;              -- rise inertial delay
                        tHL: Time);             -- fall inertial delay
                port(
                        In0  : in MVL7;
                        In1  : in MVL7;
                        Sel  : in MVL7;
                        Output: out MVL7);
        end component;


-------------------------------------------------------------------------
-- This component is the OR gate.
-------------------------------------------------------------------------
      component ORGATE                              -- ZYCAD component
          generic (N: Positive;                     -- N input OR gate
                tLH: Time;                          -- rise inertial delay
                tHL: Time);                         -- fall inertial delay
          port(input: in MVL7_VECTOR (1 to N);      -- N inputs
                output: out MVL7);                  -- one output
      end component;


-------------------------------------------------------------------------
-- This component is the NOR gate.
-------------------------------------------------------------------------
      component NORGATE                             -- ZYCAD component
          generic (N: Positive;                     -- N input NOR gate
                tLH: Time;                          -- rise inertial delay
                tHL: Time);                         -- fall inertial delay
          port(input: in MVL7_VECTOR (1 to N);      -- N inputs
                output: out MVL7);                  -- one output
      end component;
```

```
-------------------------------------------------------------------------
-- This component is the AND gate.
-------------------------------------------------------------------------
    component ANDGATE                          -- ZYCAD component
        generic (N: Positive;                  -- N input AND gate
                 tLH: Time;                    -- rise inertial delay
                 tHL: Time);                   -- fall inertial delay
        port(input: in MVL7_VECTOR (1 to N);   -- N inputs
             output: out MVL7);                -- one output
    end component;

-------------------------------------------------------------------------
-- This component is the Inverter gate.
-------------------------------------------------------------------------
    component INVGATE                          -- ZYCAD component
        generic (tLH: Time;                    -- rise inertial delay
                 tHL: Time);                   -- fall inertial delay
        port(input: in MVL7;                   -- input
             output: out MVL7);                -- output
    end component;

-------------------------------------------------------------------------
-- This component is the CHANGE_DETECTOR.
-------------------------------------------------------------------------
        component CHANGE_DETECTOR
                generic(Delay : Time);
                port   (Input : in MVL7;
                        Output: out MVL7);
        end component;

-------------------------------------------------------------------------
-- This component is the edge triggered DFF.
-------------------------------------------------------------------------
        component EDGE_TRIGGERED_DFF
                port(  D      : in MVL7;
                       RESET: in MVL7;
                       CP     : in MVL7;
                       CPnot:  in MVL7;
                       Q      : out MVL7;
                       Qnot   : out MVL7);
        end component;

begin
        -- component instantiations

        TOS_pointer: TOS_SHIFTER
                generic map(Depth)
                port map(CLEAR  => Clear_TOS,
                         CP     => TOS_CP,
                         CPnot  => TOS_CPnot,
                         Output => TOS_Output);
```

```
Word_Selector: SHIFT_REGISTER
        generic map(Depth)
        port map(In_Vector => Tags_In,
                 Sel0      => Selector,
                 Clockin   => Word_Sel_CP,
                 Clear     => OR3_out,
                 SR_Output => SR_Output);

-------------------------------------------------------------------
-- The following code generates the MUXs used to select which of the
--  word select registers (TOS_pointer or Word_Selector) to use.
-------------------------------------------------------------------
M1:
for I in Depth-1 downto 0 generate
        MUXs: MUX2x1
                generic map(MUX_Delay, MUX_Delay)
                port map(In0    => SR_Output(I),
                         In1    => TOS_Output(I),
                         Sel    => WSR_Select,
                         Output => Select_Word_Signal(I));

        Resolved_Select_Word <= Select_Word_Signal(I);

end generate;


OR1: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => Clear1,
                 input(2) => Clear2,
                 output   => Or1_out);


OR2: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => Word_Selector_CP,
                 input(2) => D_CP,
                 output   => Word_Sel_CP);


OR3: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => AND1_out,
                 input(2) => AND4_out,
                 output   => Or3_out);
```

```
OR4: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => AND2_out,
                 input(2) => AND3_out,
                 output   => OR4_out);

AND1: ANDGATE
        generic map(2,                          -- 2 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => CD2_out,
                 input(2) => Selector_not,
                 output   => AND1_out);

AND2: ANDGATE
        generic map(2,                          -- 2 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => CD3_out,
                 input(2) => NOR1_out,
                 output   => AND2_out);

AND3: ANDGATE
        generic map(2,                          -- 2 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => CD2_out,
                 input(2) => Selector,
                 output   => AND3_out);

AND4: ANDGATE
        generic map(2,                          -- 2 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => OR1_out,
                 input(2) => Selector_not,
                 output   => AND4_out);

DFF1: edge_triggered_DFF
        port map(D     => SR_Select,
                 RESET => Clear1,
                 CP    => D_CP,
                 CPnot => D_CPnot,
                 Q     => Selector,
                 Qnot  => Selector_Not);
```

```
DFF2: edge_triggered_DFF
        port map(D      => Selector,
                 RESET => Clear1,
                 CP     => OR4_out,
                 CPnot  => OR4_outnot,
                 Q      => DFF2_out,
                 Qnot   => OPEN);


CD1: CHANGE_DETECTOR
        generic map(Change_Detector_Delay)
        port map(Input  => SR_Select,
                 Output => D_CP);


CD2: CHANGE_DETECTOR
        generic map(Change_Detector_Delay)
        port map(Input  => Selector,
                 Output => CD2_out);


CD3: CHANGE_DETECTOR
        generic map(Change_Detector_Delay)
        port map(Input  => NOR1_out,
                 Output => CD3_out);


INV1: INVGATE
        generic map(Inverter_Delay, Inverter_Delay)
        port map(input  => D_CP,
                 output => D_CPnot);


INV2: INVGATE
        generic map(Inverter_Delay, Inverter_Delay)
        port map(input  => OR4_out,
                 output => OR4_outnot);


NOR1: NORGATE
        generic map(2,                          -- 2 inputs
                    NOR_Delay,                  -- rise inertial delay
                    NOR_Delay)                  -- fall inertial delay
        port map(input(1) => Resolved_Select_Word,
                 input(2) => Selector,
                 output   => NOR1_out);
```

```
    NOR2: NORGATE
          generic map(2,                    -- 2 inputs
                      NOR_Delay,            -- rise inertial delay
                      NOR_Delay)            -- fall inertial delay
          port map(input(1) => DFF2_out,
                   input(2) => Selector,
                   output   => Bit_Select);

    Select_Word <= Select_Word_Signal;
    Shifting    <= Selector;


end structure;
```

WORD_SELECT_CLOCK

```
--------------------------------------------------------------------------
-- Date:    10 September 1991
-- Version: 1.5
--
-- Filename: word_select_clock.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the
--              WORD_SELECT_CLOCK component of the CAM chip.
--              The outputs are fed into the word select registers and act
--              as the clock inputs to those registers.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc.
--                     needed for this file.
--
-- History: Version 1.1 (21 August 1991)
--          Version 1.2 (29 August 1991) - took off the CP input into AND1.
--          Version 1.3 (5 September 1991) - added a buffer to the TOS_Clock
--              port so it will transition at the same time as TOS_ClockNot.
--          Version 1.4 (5 September 1991) - The TOS_Clock and TOS_ClockNot
--              ports go into the TOS_shifter. The values are valid after
--              the falling edge of the clock and the way I had it there was
--              no falling edge. To fix that, I put in CHANGE_DETECTORs to
--              give a rising and falling edge. I wanted this to occur only
--              when a Read was high so I ANDed this signal with the results
--              of the change_detectors. Also, Master_Reset port was used to
--              clock in the Master_Reset signal but the DFFs I'm using don't
--              need a clock to reset.
--          Version 1.5 (10 September 1991) - deleted the Read input into
--              AND3 gate. As long as Counting is '1' then a Read has been
--              requested.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;

entity WORD_SELECT_CLOCK is
        port(   Read                    : in MVL7;
                Write                   : in MVL7;
                Search_Complete         : in MVL7;
                Resolved_Tags           : in MVL7;
                CP                      : in MVL7;
                Counting                : in MVL7;
                TOS_Clock               : out MVL7;
```

```
            TOS_ClockNot        : out MVL7;
            Word_Sel_Reg_Clock: out MVL7);
end WORD_SELECT_CLOCK;


architecture structure of WORD_SELECT_CLOCK is

-------------------------------------------------------------------------
-- This component is the AND gate.
-------------------------------------------------------------------------
    component ANDGATE                              -- ZYCAD component
        generic (N: Positive;                      -- N input AND gate
            tLH: Time;                             -- rise inertial delay
            tHL: Time);                            -- fall inertial delay
        port(input: in MVL7_VECTOR (1 to N); -- N inputs
            output: out MVL7);                     -- one output
    end component;


-------------------------------------------------------------------------
-- This component is the OR gate.
-------------------------------------------------------------------------
    component ORGATE                               -- ZYCAD component
        generic (N: Positive;                      -- N input OR gate
            tLH: Time;                             -- rise inertial delay
            tHL: Time);                            -- fall inertial delay
        port(input: in MVL7_VECTOR (1 to N);  -- N inputs
            output: out MVL7);                     -- one output
    end component;


-------------------------------------------------------------------------
-- This component is the Inverter gate.
-------------------------------------------------------------------------
    component INVGATE                      -- ZYCAD component
        generic (tLH: Time;                -- rise inertial delay
            tHL: Time);                    -- fall inertial delay
        port   (input: in MVL7;            -- input
            output: out MVL7);             --  output
    end component;


-------------------------------------------------------------------------
-- This component is the Buffer gate.
-------------------------------------------------------------------------
    component BUFGATE                      -- ZYCAD component
        generic (tLH: Time;                -- rise inertial delay
            tHL: Time);                    -- fall inertial delay
        port   (input: in MVL7;            -- input
            output: out MVL7);             -- output
    end component;
```

```
--------------------------------------------------------------------------
-- This component is the CHANGE_DETECTOR.
--------------------------------------------------------------------------
        component CHANGE_DETECTOR
                generic(Delay: Time);
                port   (Input : in MVL7;
                        Output: out MVL7);
        end component;


        signal RT_not          : MVL7;
        signal Read_Miss       : MVL7;
        signal Write_out       : MVL7;
        signal Read_out        : MVL7;
        signal CD1_out         : MVL7;
        signal AND4_out        : MVL7;


begin
        -- component instantiations

        INV1: INVGATE
                generic map(Inverter_Delay,          -- rise inertial delay
                            Inverter_Delay)          -- fall inertial delay
                port map(input  => Resolved_Tags,
                         output => RT_not);


        INV2: INVGATE
                generic map(Inverter_Delay,          -- rise inertial delay
                            Inverter_Delay)          -- fall inertial delay
                port map(input  => AND4_out,
                         output => TOS_ClockNot);


        BUF1: BUFGATE
                generic map(BUF_Delay,               -- rise inertial delay
                            BUF_Delay)               -- fall inertial delay
                port map(input  => AND4_out,
                         output => TOS_Clock);


        AND1: ANDGATE
                generic map(3,                                -- 3 inputs
                            AND_Delay,                        -- rise inertial delay
                            AND_Delay)                        -- fall inertial delay
                port map(input(1) => Read,
                         input(2) => Search_Complete,
                         input(3) => RT_not,
                         output   => Read_Miss);
```

```
AND2: ANDGATE
        generic map(3,                          -- 3 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => Write,
                 input(2) => Search_Complete,
                 input(3) => Resolved_Tags,
                 output   => Write_out);


AND3: ANDGATE
        generic map(2,                          -- 2 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => Counting,
                 input(2) => CP,
                 output   => Read_out);


AND4: ANDGATE
        generic map(2,                          -- 2 inputs
                    AND_Delay,                  -- rise inertial delay
                    AND_Delay)                  -- fall inertial delay
        port map(input(1) => Read_Miss,
                 input(2) => CD1_out,
                 output   => AND4_out);


OR1: ORGATE
        generic map(2,                          -- 2 inputs
                    OR_Delay,                   -- rise inertial delay
                    OR_Delay)                   -- fall inertial delay
        port map(input(1) => Write_out,
                 input(2) => Read_out,
                 output   => Word_Sel_Reg_Clock);


CD1: CHANGE_DETECTOR
        generic map(Change_Detector_Delay)
        port map(Input  => Read_Miss,
                 Output => CD1_out);

end structure;
```

Appendix D: The VHDL Code for the CAM_chip and THE_CONTROLLER

This appendix contains the VHDL code of the CAM chip and the controller. The CAM chip is the highest hierarchical level of this thesis. The controller is a component on the chip and represents all control logic needed to exercise the CAM array. The entity of the CAM chip is first presented followed by the CAM chip's structural description. Finally, the VHDL of the controller is given.

CAM_Chip Entity

```
--------------------------------------------------------------------------
-- Date:    9 September 1991
-- Version: 2.0
-- Filename: cam_chip_entity.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
-- Description: This file contains the entity of the CAM chip.
-- Associated files: chip_pkg.vhd          : This file is where the size of
--                                           the CAM array is defined. Other
--                                           declarations are also contained
--                                           in this file.
--                   chip_pkg_body.vhd     : This file contains the sub-
--                                           routines Wired_And and Wired_Or
--                                           used by the chip.
--                 cam_chip_structure.vhd: This file contains the structure
--                                           of the CAM chip. It is formed by
--                                           generating copies of the CAM
--                                           cell.
-- History: Version 1.0 (3 May 1991)
--          Version 1.1 (12 June 9191) - replaced M, RX, and RY with the actual
--              outputs of the entire chip T, R, and P.
--          Version 1.2 (25 June 1991) - changed name of outputs to correspond
--              with cam_chip_structure wire connections.
--          Version 1.3 (3 July 1991) - added port Load to allow clock into
--              chip as input into registers.
--          Version 1.4 (15 July 1991) - changed ports to correspond to
--              changing input register to Address and Data register.
--          Version 1.5 (8 August 1991) - added ports for Valid_Out bit and
--              Resolved_Tags. These are preliminary changes to get ready to
--              go in and change what I have to what I really need.
--          Version 1.6 (28 August 1991) - deleted the bit select registers
--              and I will use only the MUXs. The MUXs will feed into the
--              Bit_Select_Bus.
```

```
--          Version 1.7 (30 August 1991) - deleted the MUXs and replaced them
--              with inverters since that is all they were acting as.  Also
--              deleted registers to hold the Address_In, Data_In, and
--              Data_Out data and replaced them with buffers.
--          Version 2.0 (9 September 1991) - This version contains the
--              controller.  As a consequence of adding the controller, a few
--              ports were added and some deleted.
--
-- Author:  Curtis M. Winstead
--------------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;

entity CAM_chip is
        port(
                Data_In             : in Vector_Data_length;
                                          -- data input from data bus
                Address_In          : in Vector_Address_length;
                                          -- address input from
                                          -- address bus
                Read                : in MVL7;
                                          -- read request port
                Write               : in MVL7;
                                          -- write request port
                Data_Avail_MEM            : in MVL7;
                                          -- signifies that data is
                                          -- available from main memory
                Master_Reset        : in MVL7;
                                          -- used to put chip in
                                          -- initial state
                CP                  : in MVL7;
                CPnot               : in MVL7;
                                          -- the chips clock
                Data_Out            : out Vector_Word_length;
                                          -- data output to data bus
                Valid_Out           : out MVL7;
                                          -- resolved signal (Wired_OR)
                                          -- to determine validity of
                                          -- output data
                Data_Out_Available  : out MVL7;
                                          -- signifies that data are
                                          -- available on output
                                          -- ports on a read hit
                Read_Miss           : out MVL7;
                                          -- signifies a read miss
                Write_Miss          : out MVL7;
                                          -- signifies a write miss
                Write_Hit           : out MVL7);
                                          -- signifies a write hit
end entity;
```

CAM_Chip Structure

```
--------------------------------------------------------------------------
-- Date: 9 September 1991
-- Version: 2.0
--
-- Filename: cam_chip_structure.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the generated architecture of a
--          CAM cache chip.
-- Associated files: chip_pkg.vhd          : This file is where the size of
--                                           the CAM array is defined. Other
--                                           declarations are also contained
--                                           in this file.
--                   chip_pkg_body.vhd     : This file contains the sub-
--                                           routines Wired_And and Wired_Or
--                                           used by the chip.
--                   cam_chip_entity.vhd   : This file contains the entity
--                                           description of the CAM chip.
--
-- History: Version 1.0 (6 May 1991)
--          Version 1.1 (12 June 1991)
--          Version 1.2 (25 June 1991) - created lines (Buses) to connect
--              all inputs and outputs of each cell.
--          Version 1.3 (3 July 1991) - added register to store my Data_In,
--              Bit_Select, Data_Out, and Valid_Out values.
--          Version 1.4 (15 July 1991) - changed Data_In register to two
--              registers to hold Address and Data.
--          Version 1.5 (8 August 1991) - added ports for valid_out bit and
--              Resolved_Tags. These are preliminary changes to get ready to
--              go in and change what I have to what I really need.
--          Version 1.6 (28 August 1991) - deleted the bit select registers
--              and I will use only the MUXs. The MUXs will feed into the
--              Bit_Select_Bus.
--          Version 1.7 (30 August 1991) - deleted the MUXs and replaced them
--              with inverters since that is all they were acting as. Also
--              deleted registers to hold the Address_In, Data_In, and
--              Data_Out data and replaced them with buffers.
--          Version 2.0 (9 September 1991) - This version contains the
--              controller. As a consequence of adding the controller, a few
--              ports were added and some deleted.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------

library ZYCAD;
use ZYCAD.types.all;
use WORK.Chip_pkg.all;
```

```
architecture structure of CAM_chip is

-------------------------------------------------------------------------
-- The following signals are buses that connect each cell of the CAM array.
-------------------------------------------------------------------------
        -- The MSBs are for the address, the LSBs are for the data
        signal Data_In_Bus      : Vector_Word_length :=
                                    (Word_length-1 downto 0 => '0');
        signal Bit_Select_Bus   : Vector_Word_length :=
                                    (Word_length-1 downto 0 => '0');

        signal Word_Select_Bus : Vector_Depth := (Depth-1 downto 0 => '0');

-------------------------------------------------------------------------
-- This signal is the output vector of the XORs that evaluate the validity
-- of the outgoing data.
-------------------------------------------------------------------------
        signal Valid_Out_Signal: Vector_Data_length;


-------------------------------------------------------------------------
-- This signal is the converted Resolved_Signal_Tag from Wired_Or_Vector to
--  MVL7_Vector.
-------------------------------------------------------------------------
        signal Tag_Vector: Vector_Depth;


-------------------------------------------------------------------------
-- The following signals are resolved signals. The subtypes are defined in
-- chip_pkg.vhd. The functions are defined in chip_pkg_body.vhd.
-------------------------------------------------------------------------
        signal Resolved_Signal_Data_Out   : Wired_Or_Vector(
                                              Word_length-1 downto 0);
        signal Resolved_Signal_Data_Check: Wired_Or_Vector(
                                              Data_length-1 downto 0);
        signal Resolved_Signal_Tag        : Wired_And_Vector(Depth-1 downto 0);
        signal Resolved_Tags              : Wired_Or_Type;
        signal Resolved_Signal_Valid_Out  : Wired_And_Type;

-------------------------------------------------------------------------
-- This signal holds the data out signals that are converted from Wired_Or
-- type to MVL7 vector before going into the Data_Out_Buffers.
-------------------------------------------------------------------------
        signal Data_Out_Vector: Vector_Word_length;


-------------------------------------------------------------------------
-- This signal contains the data from the Data_Out_Buffers.  It connects to
-- the Data_Out ports of the chip and the controller.
-------------------------------------------------------------------------
        signal Data_Out_Signal_Vector: Vector_Word_length;
```

```
-------------------------------------------------------------------------
-- This signal connects the controllers Select_Word port to the OR gates that
--  feed the Word_Select_Bus.
-------------------------------------------------------------------------
        signal Word_Select: Vector_Depth;


-------------------------------------------------------------------------
-- This signal is connected to the output port of the controller.  It is
--  connected to the Bit_Select_Bus.
-------------------------------------------------------------------------
        signal Bit_Select_Signal: MVL7;


-------------------------------------------------------------------------
-- This is the basic component of the CAM chip.  These cells will be put into
--  an array to form the CAM chip array.
-------------------------------------------------------------------------
        component CAM_cell
                port(
                        D:  in MVL7;            -- data line
                        B:  in MVL7;            -- bit select line
                        W:  in MVL7;            -- word select line
                        M:  out MVL7;           -- match line
                        RY: out MVL7;           -- data output (W and C)
                        RX: out MVL7);          -- data output (W and Cnot)
        end component;


-------------------------------------------------------------------------
-- This component is the exclusive-or gate.
-------------------------------------------------------------------------
        component XORGATE                                   -- ZYCAD component
                generic (N: Positive;                       -- N input XOR gate
                        tLH: Time;                          -- rise inertial delay
                        tHL: Time);                         -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                        output: out MVL7);                  -- one output
        end component;


-------------------------------------------------------------------------
-- This component is the OR gate.
-------------------------------------------------------------------------
        component ORGATE                                    -- ZYCAD component
                generic (N: Positive;                       -- N input OR gate
                        tLH: Time;                          -- rise inertial delay
                        tHL: Time);                         -- fall inertial delay
                port(input: in MVL7_VECTOR (1 to N);        -- N inputs
                        output: out MVL7);                  -- one output
        end component;
```

```
-------------------------------------------------------------------------
-- This component is the buffer gate.
-------------------------------------------------------------------------
        component BUFGATE                       -- ZYCAD component
                generic (tLH: Time;             -- rise inertial delay
                         tHL: Time);            -- fall inertial delay
                port(input: in MVL7;            -- input
                     output: out MVL7);         -- output
        end component;

-------------------------------------------------------------------------
-- This component is the controller. The file is called 'the_controller'.
-------------------------------------------------------------------------
        component THE_CONTROLLER
                port(   Read                    : in MVL7;
                        Write                   : in MVL7;
                        Resolved_Tags           : in MVL7;
                        CP                      : in MVL7;
                        CPnot                   : in MVL7;
                        Data_Avail_MEM          : in MVL7;
                        Master_Reset            : in MVL7;
                        Data_Out                : in Vector_Word_length;
                        Data_In                 : in Vector_Address_length;
                        Resolved_Signal_Tag: in Vector_Depth;
                        Data_Out_Available  : out MVL7;
                        Read_Miss           : out MVL7;
                        Write_Miss          : out MVL7;
                        Write_Hit           : out MVL7;
                        Bit_Select          : out MVL7;
                        Select_Word         : out Vector_Depth);
        end component;

begin
-------------------------------------------------------------------------
-- The following code automatically generates the CAM array for the address.
-- Word_length, Data_length, and Depth are defined in chip_pkg.vhd. The
-- left-most bit is the most significant bit while the right-most is the
-- least significant bit.
-------------------------------------------------------------------------
        A1:
        for I in Word_length-1 downto Data_length generate  -- Address length
                D1:
                for J in Depth-1 downto 0 generate
                        Address_Array:  CAM_cell
                                port map(
                                        D   => Data_In_Bus(I),
                                        B   => Bit_Select_Bus(I),
                                        W   => Word_Select_Bus(J),
                                        M   => Resolved_Signal_Tag(J),
                                        RY => Resolved_Signal_Data_Out(I),
                                        RX => OPEN);
```

```
            ------------------------------------------------------------
            -- This signal is a single bit that signifies whether a match
            --  on any  word has occurred.
            ------------------------------------------------------------
                    Resolved_Tags <= Resolved_Signal_Tag(J);

            end generate;

        ------------------------------------------------------------------
        -- These buffers buffer the incoming address.
        ------------------------------------------------------------------
        Address_Buffers: BUFGATE
                generic map(BUF_Delay, BUF_Delay)
                port map(input  => Address_In(I),
                         output => Data_In_Bus(I));

        end generate;

------------------------------------------------------------------------
-- The following code automatically generates the CAM array for the data.
-- Data_length, and Depth are defined in chip_pkg.vhd.  The
-- left-most bit is the most significant bit while the right-most is the
-- least significant bit (zero).
------------------------------------------------------------------------
        A2:
        for K in Data_length-1 downto 0 generate
                D1:
                for L in Depth-1 downto 0 generate
                        Data_Array:  CAM_cell
                                port map(
                                        D   => Data_In_Bus(K),
                                        B   => Bit_Select_Bus(K),
                                        W   => Word_Select_Bus(L),
                                        M   => OPEN,
                                        RY => Resolved_Signal_Data_Out(K),
                                        RX => Resolved_Signal_Data_Check(K));
                end generate;

        ------------------------------------------------------------------
        -- This code generates an array of xor gates that evaluates the
        --  Valid_Out for each bit of the data.  It xor's the Data_Out
        --  and the Data_Check bit.  Valid_Out is '1' if the bit is valid.
        ------------------------------------------------------------------
                XOR_Array:  XORGATE
                        generic map (2,                 -- 2 inputs,
                                XOR_Delay,              -- rise inertial delay,
                                XOR_Delay)              -- fall inertal delay
                        port map(
                                input(1) => Resolved_Signal_Data_Out(K),
                                input(2) => Resolved_Signal_Data_Check(K),
                                output   => Valid_Out_Signal(K));
```

```
        ------------------------------------------------------------------
        -- The port Resolved_Sigal_Valid_Out is determined by Wire-ORing the
        --  Valid_Out_Signal.
        ------------------------------------------------------------------
        Resolved_Signal_Valid_Out <= Valid_Out_Signal(K);


        ------------------------------------------------------------------
        -- These buffers buffer the incoming data.
        ------------------------------------------------------------------
        Data_Buffers: BUFGATE
                generic map(BUF_Delay, BUF_Delay)
                port map(input  => Data_In(K),
                         output => Data_In_Bus(K));


        end generate;


        ------------------------------------------------------------------
        -- This line converts the Wired_And signal to MVL7 to output onto
        --  the Valid_Out port.  (added 21 Oct 91)
        ------------------------------------------------------------------
        Valid_Out <= Wired_And_To_MVL7(Resolved_Signal_Valid_Out);


------------------------------------------------------------------------
-- The following code generates OR gates that select all cells upon resetting
--  the chip. This eneables all cells to be intialized to '0' when Master_
--  Reset goes high.
------------------------------------------------------------------------
        O1:
        for M in Depth-1 downto 0 generate
                ORs:  ORGATE
                        generic map (2,                 -- 2 inputs,
                                OR_Delay,               -- rise inertial delay,
                                OR_Delay)               -- fall inertal delay
                        port map(
                                input(1) => Master_Reset,
                                input(2) => Word_Select(M),
                                output   => Word_Select_Bus(M));
        end generate;



------------------------------------------------------------------------
-- The following code instantiates a particular cell of the array.  It allows
--  you to 'cd' into the cell for troubleshooting.  It is a copy of a cell
--  generated with the code above.  This cell is a data cell.
-- THIS CELL SHOULD NOT BE IMPLEMENTED IN HARDWARE!
------------------------------------------------------------------------
```

```
        Cell_00: CAM_cell
                port map(
                        D  => Data_In_Bus(0),                   -- across
                        B  => Bit_Select_Bus(0),                -- across
                        W  => Word_Select_Bus(0),               -- down
                        M  => Resolved_Signal_Tag(0),           -- down
                        RY => Resolved_Signal_Data_Out(0),      -- across
                        RX => Resolved_Signal_Data_Check(0));   -- across

-------------------------------------------------------------------------
-- The following code instantiates a particular cell of the array.  It allows
-- you to 'cd' into the cell for troubleshooting.  It is a copy of a cell
-- generated with the code above.  This cell is an address cell.
-- THIS CELL SHOULD NOT BE IMPLEMENTED IN HARDWARE!
-------------------------------------------------------------------------
        Cell_Address:  CAM_cell
                port map(
                        D  => Data_In_Bus(Word_length-1),                   -- across
                        B  => Bit_Select_Bus(Word_length-1),                -- across
                        W  => Word_Select_Bus(Depth-2),                     -- down
                        M  => Resolved_Signal_Tag(Depth-2),                 -- down
                        RY => Resolved_Signal_Data_Out(Word_length-1),-- across
                        RX => OPEN);                                        -- across


-------------------------------------------------------------------------
-- This code generates the buffers that buffer the output data.
-------------------------------------------------------------------------
        B1:
        for N in Word_length-1 downto 0 generate
                Data_Out_Buffers: BUFGATE
                        generic map(BUF_Delay, BUF_Delay)
                        port map(input  => Data_Out_Vector(N),
                                 output => Data_Out_Signal_Vector(N));
        end generate;


-------------------------------------------------------------------------
-- This line converts the Resolved_Signal_Data_Out and Resolved_Signal_Tag
-- signals into an MVL7 vector.
-------------------------------------------------------------------------
        Data_Out_Vector <= Wired_Or_To_MVL7_Vector(Resolved_Signal_Data_Out);
        Tag_Vector      <= Wired_And_To_MVL7_Vector(Resolved_Signal_Tag);

-------------------------------------------------------------------------
-- The following assigns the Data_Out port of the chip with the Data_Out_
-- Signal_Vector that comes out of the Data_Out_Buffers.
-------------------------------------------------------------------------
        Data_Out <= Data_Out_Signal_Vector;
```

```
-------------------------------------------------------------------------
-- This connects the controller to the rest of the chip.
-------------------------------------------------------------------------
        Controller: THE_CONTROLLER
                port map(
                        Read                    => Read,
                        Write                   => Write,
                        Resolved_Tags           => Resolved_Tags,
                        CP                      => CP,
                        CPnot                   => CPnot,
                        Data_Avail_MEM          => Data_Avail_MEM,
                        Master_Reset            => Master_Reset,
                        Data_Out                => Data_Out_Signal_Vector,
                        Data_In                 => Data_In_Bus
                                        (Word_length-1 downto Data_length),
                        Resolved_Signal_Tag => Tag_Vector,
                        Data_Out_Available  => Data_Out_Available,
                        Read_Miss               => Read_Miss,
                        Write_Miss              => Write_Miss,
                        Write_Hit               => Write_Hit,
                        Bit_Select              => Bit_Select_Signal,
                        Select_Word             => Word_Select);


-------------------------------------------------------------------------
-- This process simply connects the Bit_Select_Signal from the controller to
--  each line fo the Bit_Select_Bus.
-------------------------------------------------------------------------
        P1:
        process(Bit_Select_Signal)
          begin
                for O in Word_length-1 downto 0 loop
                        Bit_Select_Bus(O) <= Bit_Select_Signal;
                end loop;
        end process;


end structure;
```

CAM Chip Behavior

```
--------------------------------------------------------------------------
-- Date: 21 October 1991
-- Version: 1.0
--
-- Filename: cam_chip_behavior.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the architecture behavior of the
--              CAM cache chip.
-- Associated files: chip_pkg.vhd        : This file is where the size of
--                                         the CAM array is defined. Other
--                                         declarations are also contained
--                                         in this file.
--                   chip_pkg_body.vhd   : This file contains the sub-
--                                         routines Wired_And and Wired_Or
--                                         used by the chip.
--                   cam_chip_entity.vhd : This file contains the entity
--                                         description of the CAM chip.
-- History:
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.Chip_pkg.all;

architecture behavior of cam_chip is

        ------------------------------------------------------------------
        -- The following are declared types for the behavioral description.
        --  MEM_TYPE is for the CAM array State is the state types.
        ------------------------------------------------------------------
        type MEM_TYPE is array(Depth downto 1) of Vector_Word_length;
        type State is (Start_State, Read_Hit_State, Read_Miss_State,
                                    Write_Hit_State, Write_Miss_State);

        ------------------------------------------------------------------
        -- The following are constant time declarations.
        ------------------------------------------------------------------
        constant Search_Delay                : Time := 16 ns;
        constant State_Output_Delay          : Time := 3 ns;
        constant MEM_Delay                   : Time := 10 ns;
        constant Write_Delay                 : Time := 4 ns;
        constant Write_Miss_Delay            : Time := 25 ns;
        constant Data_Out_Avail_Init_Delay   : Time := 12 ns;
        constant Data_Out_Init_Delay         : Time := 16 ns;
        constant Valid_Out_Init_Delay        : Time := 19 ns;
        constant Function_Change_Delay       : Time := 3 ns;
```

```
------------------------------------------------------------------
-- This signal is used when all zeros are needed for a word.
------------------------------------------------------------------
signal All_Zeros : Vector_Word_length :=
                                (Word_length-1 downto 0 => '0');

------------------------------------------------------------------
-- This signal holds the state in which the chip is in.
------------------------------------------------------------------
signal State_Register: State := Start_State;

------------------------------------------------------------------
-- This signal holds the position of the word found in the CAM.
------------------------------------------------------------------
signal Tag : Natural := 0;

------------------------------------------------------------------
-- This signal signifies if a search hit has occurred.
------------------------------------------------------------------
signal Search_Hit: MVL7 := '0';

------------------------------------------------------------------
-- The following signals are used to multiplex the correct value.
--  The MUX code is at the end of the behavioral description.
------------------------------------------------------------------
signal cam_array, cam_arrayI, cam_arrayS, cam_arrayRH,
        cam_arrayRM, cam_arrayWH: MEM_TYPE;
signal State_RegisterS, State_RegisterRH, State_RegisterRM,
        State_RegisterWH, State_RegisterWM: State;
signal Data_Out_AvailableI, Data_Out_AvailableRH,
                        Data_Out_AvailableTEMP: MVL7;
signal Read_MissI, Read_MissRM, Read_MissTEMP: MVL7;
signal Write_HitI, Write_HitWH, Write_HitTEMP: MVL7;
signal Write_MissI, Write_MissWM, Write_MissTEMP: MVL7;
signal Data_OutI, Data_OutRH, Data_OutTEMP: MVL7_Vector
                                        (Word_length-1 downto 0);
signal Valid_OutI, Valid_OutRH, Valid_OutTEMP: MVL7;


  begin

------------------------------------------------------------------
-- Initialize:  This block is used to initialize the chip.
------------------------------------------------------------------
        Init: block(Master_Reset='1' and not Master_Reset'Stable)
        begin

                Data_Out_AvailableI <= '0' after Data_Out_Avail_Init_Delay;
                Read_MissI          <= '0' after State_Output_Delay;
                Write_HitI          <= '0' after State_Output_Delay;
                Write_MissI         <= '0' after State_Output_Delay;
```

```
            Data_OutI  <= All_Zeros after Data_Out_Init_Delay;
            Valid_OutI <= '0' after Valid_Out_Init_Delay;


            ------------------------------------------------------------
            -- This process clears the CAM array.
            ------------------------------------------------------------
            process(guard)
            begin

                  for I in Depth downto 1 loop
                        cam_arrayI(I) <= All_Zeros;
                  end loop;

            end process;

      end block Init;


-------------------------------------------------------------------------
-- Search state:  In this state the address is searched for and the state
-- register is assigned a value.
-------------------------------------------------------------------------
      Search_process: process
      begin
            wait on Read, Write;
            if Read = '1' or Write = '1' then

                  ------------------------------------------------
                  -- This loop searches the CAM array and marks the
                  --  found word with Tag.
                  ------------------------------------------------
                  Search_loop:
                  for J in Depth downto 1 loop

                        if (cam_array(J)
                              (Word_length-1 downto Data_length)
                                          = Address_In) then
                              Tag <= J;
                              exit;
                        else Tag <= 0;
                        end if;

                  end loop;


                  wait for Search_Delay;
```

```
                    ------------------------------------------------
                    -- This 'if' statement sets the state register to
                    --  the proper state value.
                    ------------------------------------------------
                    if Read = '1' and Tag /= 0 then
                            State_RegisterS <= Read_Hit_State;
                    elsif Read = '1' and Tag = 0 then
                            State_RegisterS <= Read_Miss_State;
                    elsif Write = '1' and Tag /= 0 then
                            State_RegisterS <= Write_Hit_State ;
                    else
                            State_RegisterS <= Write_Miss_State;
                    end if;

            end if;

    end process Search_process;


-------------------------------------------------------------------------
-- Read Hit state.
-------------------------------------------------------------------------
    RH: process
            variable Kcount: Natural;  -- this variable is used for
                                       --  for troubleshooting.
    begin

    wait on State_Register until State_Register = Read_Hit_State;

            ------------------------------------------------------
            -- This is the prefetch loop.
            ------------------------------------------------------
            for K in Tag-1 to Prefetch_Block_Size+Tag-2 loop

            Kcount := K;
                    ------------------------------------------------
                    -- This is the first word being read from the CAM.
                    ------------------------------------------------
                    if K = Tag-1 then
                            wait for 40 ns;
                            Data_OutRH <= cam_array((K mod Depth) + 1);
                            wait for AND_Delay;
                            Valid_OutRH <= '1';
                            wait for OR_Delay;
                            Data_Out_AvailableRH <= '1';
                            wait for Change_Detector_Delay;
                            Data_Out_AvailableRH <= '0';
                            wait for 2 ns;
                    end if;
```

```
                    ------------------------------------------------
                    -- These are the remaining words being read from CAM.
                    ------------------------------------------------
                    if K /= Tag-1 then
                        wait for Clock_Period - AND_Delay -
                                OR_Delay - Change_Detector_Delay;
                        Data_OutRH <= cam_array((K mod Depth) + 1);
                        wait for AND_Delay + OR_Delay;
                        Data_Out_AvailableRH <= '1';
                        wait for Change_Detector_Delay;
                        Data_Out_AvailableRH <= '0';
                    end if;

                end loop;

                    ------------------------------------------------
                    -- This signifies the end of the prefetch cycle.
                    ------------------------------------------------
                    wait for 13 ns - AND_Delay -
                                OR_Delay - Change_Detector_Delay;
                    Data_OutRH <= All_Zeros;
                    wait for 3 ns;
                    Valid_OutRH <= '0';

                State_RegisterRH <= Start_State;  -- reset state register

        end process RH;

-------------------------------------------------------------------------
-- Read Miss state. In this state the CAM waits for data from main memory
--  before writing it into the CAM.
-------------------------------------------------------------------------
        RM: process
                variable TOS_pointer : Positive := 1;
        begin

        wait on State_Register until State_Register = Read_Miss_State;

        ------------------------------------------------------------
        -- This assignment copies the CAM array into the local process.
        ------------------------------------------------------------
        cam_arrayRM <= cam_array;

                wait for State_Output_Delay;
                Read_MissRM <= '1';
                TOS_pointer := (TOS_pointer mod Depth) + 1;

                wait for MEM_Delay;
                cam_arrayRM(TOS_pointer)(Word_length-1 downto Data_length)
                        <= Address_In;  -- write address
                cam_arrayRM(TOS_pointer)(Data_length-1 downto 0)
```

```
                    <= Data_In;    -- write data

                Read_MissRM <= '0';
                State_RegisterRM <= Start_State;  -- reset state register

        end process RM;

-------------------------------------------------------------------------
-- Write Hit state.  Data are written over the old data.
-------------------------------------------------------------------------
        WH: process
        begin

        wait on State_Register until State_Register = Write_Hit_State:

        ---------------------------------------------------------------
        -- This assignment copies the CAM array into the local process.
        ---------------------------------------------------------------
        cam_arrayWH <= cam_array;

                wait for State_Output_Delay;
                Write_HitWH <= '1';

                wait for Write_Delay;
                cam_arrayWH(Tag)(Word_length-1 downto Data_length)
                        <= Address_In;          -- write address
                cam_arrayWH(Tag)(Data_length-1 downto 0)
                        <= Data_In;             -- write data

                Write_HitWH <= '0';

                State_RegisterWH <= Start_State;  -- reset state register

        end process WH;

-------------------------------------------------------------------------
-- Write Miss state.
-------------------------------------------------------------------------
        WM: process
        begin

        wait on State_Register until State_Register = Write_Miss_State;
                wait for State_Output_Delay + 1 ns;
                Write_MissWM <= '1';
                wait for Function_Change_Delay;
                Write_MissWM <= '0';

                wait for Write_Miss_Delay;
                State_RegisterWM <= Start_State;  -- reset state register

        end process WM;
```

```
-------------------------------------------------------------------------
-- The following code are multiplexed values. The TEMP signals are
-- necessary to avoid having more than one source for the signals.
-------------------------------------------------------------------------
        cam_array <= cam_arrayI when not cam_arrayI'quiet else
                     cam_arrayS when not cam_arrayS'quiet else
                     cam_arrayRH when not cam_arrayRH'quiet else
                     cam_arrayRM when not cam_arrayRM'quiet else
                     cam_arrayWH when not cam_arrayWH'quiet else
                     cam_array;

        State_Register <= State_RegisterS when not State_RegisterS'quiet else
                     State_RegisterRH when not State_RegisterRH'quiet else
                     State_RegisterRM when not State_RegisterRM'quiet else
                     State_RegisterWH when not State_RegisterWH'quiet else
                     State_Register;

        Data_Out_AvailableTEMP <= Data_Out_AvailableRH when
                                      not Data_Out_AvailableRH'quiet else
                     Data_Out_AvailableI when
                                      not Data_Out_AvailableI'quiet else
                     Data_Out_AvailableTEMP;
        Data_Out_Available <= Data_Out_AvailableTEMP;

        Read_MissTEMP <= Read_MissRM when not Read_MissRM'quiet else
                     Read_MissI when not Read_MissI'quiet else
                     Read_MissTEMP;
        Read_Miss <= Read_MissTEMP;

        Write_HitTEMP <= Write_HitWH when not Write_HitWH'quiet else
                     Write_HitI when not Write_HitI'quiet else
                     Write_HitTEMP;
        Write_Hit <= Write_HitTEMP;

        Write_MissTEMP <= Write_MissWM when not Write_MissWM'quiet else
                     Write_MissI when not Write_MissI'quiet else
                     Write_MissTEMP;
        Write_Miss <= Write_MissTEMP;

        Data_OutTEMP <= Data_OutRH when not Data_OutRH'quiet else
                     Data_OutI when not Data_OutI'quiet else
                     Data_OutTEMP;
        Data_Out <= Data_OutTEMP;

        Valid_OutTEMP <= Valid_OutRH when not Valid_OutRH'quiet else
                     Valid_OutI when not Valid_OutI'quiet else
                     Valid_OutTEMP;
        Valid_Out <= Valid_OutTEMP;


end behavior;
```

THE_CONTROLLER

```
--------------------------------------------------------------------------
-- Date:    12 September 1991
-- Version: 1.2
--
-- Filename: the_controller.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the entity and structure of the
--              chip controller. It provides the inputs to the CAM array.
--
-- Associated files:
--      chip_pkg.vhd : This file contains constants, variables, etc. needed
--                     for this file.
--
-- History: Version 1.0 (30 August 1991)
--          Version 1.1 (10 September 1991) - added a port to word_select
--              component (Bit_Select) so port on controller also had to be
--              added.
--          Version 1.2 (12 September 1991) - deleted port Counting since
--              Bit_Select does essentially the same thing.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.chip_pkg.all;

entity THE_CONTROLLER is
        port(   Read                    : in MVL7;
                Write                   : in MVL7;
                Resolved_Tags           : in MVL7;
                CP                      : in MVL7;
                CPnot                   : in MVL7;
                Data_Avail_MEM          : in MVL7;
                Master_Reset            : in MVL7;
                Data_Out                : in Vector_Word_length;
                Data_In                 : in Vector_Address_length;
                Resolved_Signal_Tag     : in Vector_Depth;
                Data_Out_Available      : out MVL7;
                Read_Miss               : out MVL7;
                Write_Miss              : out MVL7;
                Write_Hit               : out MVL7;
                Bit_Select              : out MVL7;
                Select_Word             : out Vector_Depth);
end THE_CONTROLLER;

architecture structure of THE_CONTROLLER is
```

```
component FUNCTION_CHANGE_DETECTOR
        port(   Read            : in MVL7;
                Write           : in MVL7;
                Function_Change: out MVL7);
end component;

component OPERATION_STATUS
        port(
                Data                    : in Vector_Word_length;
                Prefetching             : in MVL7;
                Read                    : in MVL7;
                Write                   : in MVL7;
                Resolved_Tags           : in MVL7;
                Search_Complete         : in MVL7;
                Data_Out_Available      : out MVL7;   -- read hit
                Read_Miss               : out MVL7;   -- read miss
                Write_Miss              : out MVL7;   -- write miss
                Write_Hit               : out MVL7); -- write hit
end component;

component PREFETCH_STATUS
        port(   CP              : in MVL7;
                CPnot           : in MVL7;
                Read            : in MVL7;
                Resolved_Tags   : in MVL7;
                Search_Complete: in MVL7;
                Reset           : in MVL7;
                Counting        : out MVL7);
end component;

component SEARCH_STATUS
        port(
                Data            : in Vector_Address_length;
                Read            : in MVL7;
                Write           : in MVL7;
                Function_Change : in MVL7;
                Reset_DFF       : in MVL7;
                Counting        : in MVL7;
                Search_Complete : out MVL7);
end component;

component SELECT_WORD_SELECT
        port(   Read            : in MVL7;
                Resolved_Tags   : in MVL7;
                Search_Complete : in MVL7;
                Function_Change : in MVL7;
                Master_Reset    : in MVL7;
                Data_Avail_MEM: in MVL7;
                WSR_Select      : out MVL7);
end component;
```

```
component WORD_SELECT
        port(
                Clear_TOS          : in MVL7;
                TOS_CP             : in MVL7;
                TOS_CPnot          : in MVL7;
                Word_Selector_CP: in MVL7;
                WSR_Select         : in MVL7;
                Tags_In            : in Vector_Depth;
                Clear1             : in MVL7;
                Clear2             : in MVL7;
                SR_Select          : in MVL7;
                Shifting           : out MVL7;
                Bit_Select         : out MVL7;
                Select_Word        : out Vector_Depth);
end component;


component WORD_SELECT_CLOCK
        port(   Read                    : in MVL7;
                Write                   : in MVL7;
                Search_Complete         : in MVL7;
                Resolved_Tags           : in MVL7;
                CP                      : in MVL7;
                Counting                : in MVL7;
                TOS_Clock               : out MVL7;
                TOS_ClockNot            : out MVL7;
                Word_Sel_Reg_Clock: out MVL7);
end component;


signal Function_Change      : MVL7;
signal WSR_Sel              : MVL7;
signal Search_Complete      : MVL7;
signal Counting_Signal      : MVL7;
signal TOS_CP               : MVL7;
signal TOS_CPnot            : MVL7;
signal Word_Selector_CP     : MVL7;
signal Shifting_Signal      : MVL7;

begin


Detect_Function_Change: FUNCTION_CHANGE_DETECTOR
        port map(Read           => Read,
                 Write          => Write,
                 Function_Change => Function_Change);
```

```
Op_Stat: OPERATION_STATUS
        port map(
                Data                => Data_Out,
                Prefetching         => Shifting_Signal,
                Read                => Read,
                Write               => Write,
                Resolved_Tags       => Resolved_Tags,
                Search_Complete     => Search_Complete,
                Data_Out_Available  => Data_Out_Available,
                Read_Miss           => Read_Miss,
                Write_Miss          => Write_Miss,
                Write_Hit           => Write_Hit);


Prefetch: PREFETCH_STATUS
        port map(
                CP              => CP,
                CPnot           => CPnot,
                Read            => Read,
                Resolved_Tags   => Resolved_Tags,
                Search_Complete => Search_Complete,
                Reset           => Master_Reset,
                Counting        => Counting_Signal);


Search_Stat: SEARCH_STATUS
        port map(
                Data            => Data_In,
                Read            => Read,
                Write           => Write,
                Function_Change => Function_Change,
                Reset_DFF       => Master_Reset,
                Counting        => Counting_Signal,
                Search_Complete => Search_Complete);


Sel_Word_Sel: SELECT_WORD_SELECT
        port map(
                Read            => Read,
                Resolved_Tags   => Resolved_Tags,
                Search_Complete => Search_Complete,
                Function_Change => Function_Change,
                Master_Reset    => Master_Reset,
                Data_Avail_MEM  => Data_Avail_MEM,
                WSR_Select      => WSR_Sel);
```

```
    Word_Sel: WORD_SELECT
        port map(
            Clear_TOS           => Master_Reset,
            TOS_CP              => TOS_CP,
            TOS_CPnot           => TOS_CPnot,
            Word_Selector_CP    => Word_Selector_CP,
            WSR_Select          => WSR_Sel,
            Tags_In             => Resolved_Signal_Tag,
            Clear1              => Master_Reset,
            Clear2              => Function_Change,
            SR_Select           => Counting_Signal,
            Shifting            => Shifting_Signal,
            Bit_Select          => Bit_Select,
            Select_Word         => Select_Word);


    Word_Sel_CLock: WORD_SELECT_CLOCK
        port map(
            Read                => Read,
            Write               => Write,
            Search_Complete     => Search_Complete,
            Resolved_Tags       => Resolved_Tags,
            CP                  => CP,
            Counting            => Counting_Signal,
            TOS_Clock           => TOS_CP,
            TOS_ClockNot        => TOS_CPnot,
            Word_Sel_Reg_Clock => Word_Selector_CP);

end structure;
```

Appendix E: Chip_pkg and dual_phase_clock

This appendix contains the chip_pkg.vhd, chip_pkg_body.vhd, and dual_phase_clock.vhd files. The file chip_pkg.vhd contains declarations for constants, types, and functions. The file chip_pkg_body.vhd contains the wired-OR, wired-AND, and conversion functions.

Chip_pkg

```
-------------------------------------------------------------------
-- Date:    9 September 1991
-- Version:  1.5
--
-- Filename:  chip_pkg.vhd
-- System:    ZYCAD, VLSI net
-- Language:  VHDL
--
-- Description:  This file contains the package of the CAM chip. This is the
--               file used to define the size of the CAM array. This file
--               also contains other declarations.
-- Associated files:  cam_cell_entity.vhd     : This file contains the entity
--                                              description of the CAM cell.
--                    cam_cell_structure.vhd : This file contains the gate level
--                                              design of the CAM cell.
--                    chip_pkg_body.vhd      : This file contains the sub-
--                                              routines Wired_And and Wired_Or
--                                              used by the chip.
--                    cam_chip_entity.vhd    : This file contains the entity
--                                              description of the CAM chip.
--                    cam_chip_structure.vhd: This file contains the structure
--                                              of the CAM chip. It is formed by
--                                              generating copies of the CAM
--                                              cell.
--                    chip_stimulus.vhd      : This file exercises the chip and
--                                              provides inputs to test the chip.
--                    chip_test_bench.vhd    : This file contains the test bench
--                                              for the CAM chip.
--                    chip_config.vhd        : This file contains the
--                                              configuration of the system.
--                    clock.vhd              : You guess!
--
-- History:  Version 1.0 (6 May 1991)
--           Version 1.1 (12 June 1991) - added Vector_Word_length and
--               Vector_Depth.
```

```
--          Version 1.2 (13 June 1991) - added function declarations for
--              Wired_Or and Wired_And as well as the various types used to
--              support them.
--          Version 1.3 (3 July 1991) - added funtion declaratation for
--              Wired_Or_To_MVL7_Vector.
--          Version 1.4 (15 July 1991) - changed Word_length to include
--              Data_length + Address_length.
--          Version 1.5 (9 September 1991) - added function declaration for
--              Wired_And_To_MVL7_Vector.
--
-- Author: Curtis M. Winstead
-------------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;


package Chip_pkg is
-------------------------------------------------------------------------------
-- The following constants are used to set the size of the CAM array.
-- Set Address_length to the desired length of the address in the CAM array.
-- Set Data_length to the desired length of the data in the CAM array.
-- Word_length is the sum of the address and data length.
-- Set Depth to the desired depth of the CAM array.
-------------------------------------------------------------------------------
        constant Address_length: Positive := 3;
        constant Data_length    : Positive := 3;
        constant Word_length    : Positive := Address_length + Data_length;

        constant Depth          : Positive := 3;

-------------------------------------------------------------------------------
-- This is the number of bits in the pretech counter. Set this number to
-- determine how large the prefetch block size can be. For example, an '8'
-- will allow the counter to prefetch 256 lines.
-------------------------------------------------------------------------------
        constant Bits_in_Counter: Positive := 5;


-------------------------------------------------------------------------------
-- This is the integer value for the prefetch block size. Set this number
-- to the desired number of lines to prefetch on a Read Hit. This number
-- is relates directly to Bits_in_Counter above. This number must be less
-- than or equal to 2**(Bits_in_Counter)-1.
-------------------------------------------------------------------------------
        constant Prefetch_Block_Size: Positive := 25;


-------------------------------------------------------------------------------
-- This is the clock period used in the chip and all other related components.
-------------------------------------------------------------------------------
        constant Clock_Period: Time := 30 ns;
```

```
-------------------------------------------------------------------------
-- The following types define the various inputs and outputs of the CAM
-- chip.
-------------------------------------------------------------------------
        subtype Vector_Word_length is MVL7_Vector(
                Word_length-1 downto 0);

        subtype Vector_Address_length is MVL7_Vector(
                Word_length-1 downto Data_length);          -- MSBs

        subtype Vector_Data_length is MVL7_Vector(
                Data_length-1 downto 0);                    -- LSBs

        subtype Vector_Depth is MVL7_Vector(
                Depth-1 downto 0);

-------------------------------------------------------------------------
-- This is an unconstrained array that supports the resolution functions below.
-------------------------------------------------------------------------
        type Unconstrained_Vector is array(
                                Integer range<>) of MVL7;

-------------------------------------------------------------------------
-- Below are the declarations for the resolution functions Wired_Or and
--  Wired_And.  The functions can be found in chip_pkg_body.vhd.
-------------------------------------------------------------------------
        function Wired_Or  (Input:  Unconstrained_Vector) return MVL7;
        function Wired_And (Input:  Unconstrained_Vector) return MVL7;

-------------------------------------------------------------------------
-- The following types are arrays that hold the resolved signals.
--  They are unconstrained arrays that are constrained when instantiated for
--  a particular purpose.
-------------------------------------------------------------------------
        subtype Wired_Or_Type is Wired_Or MVL7;
        type Wired_Or_Vector is array (Integer range <>) of Wired_Or_Type;

        subtype Wired_And_Type is Wired_And MVL7;
        type Wired_And_Vector is array (Integer range <>) of Wired_And_Type;

-------------------------------------------------------------------------
-- The following are the declarations of the conversion functions.
-------------------------------------------------------------------------
        function Wired_Or_To_MVL7_Vector (Input:  Wired_Or_Vector)
                                        return Vector_Word_length;

        function Wired_And_To_MVL7_Vector (Input:  Wired_And_Vector)
                                        return Vector_Depth;
        function Wired_And_To_MVL7 (Input:  Wired_And_Type) return MVL7;
```

```
-------------------------------------------------------------------------
-- The following are gate delays used by the generic statements.
-------------------------------------------------------------------------
        constant Inverter_Delay     : Time := 1 ns;
        constant BUF_Delay          : Time := 1 ns;
        constant And_Delay          : Time := 3 ns;
        constant OR_Delay           : Time := 4 ns;
        constant NAND_Delay         : Time := 2 ns;
        constant NOR_Delay          : Time := 3 ns;
        constant XOR_Delay          : Time := 4 ns;
        constant XNOR_Delay         : Time := 4 ns;
        constant DFF_Delay          : Time := 6 ns;
        constant MUX_Delay          : Time := 2 ns;


-------------------------------------------------------------------------
-- This constant is the amount of time the CHANGE_DETECTOR
--  will produce  a '1' after the signal changes.  Set this constant equal
--  to the time the signal should be high upon a signal change.
-------------------------------------------------------------------------
        constant Change_Detector_Delay: Time := 5 ns;

end Chip_pkg;
```

Chip_pkg_body

```
--------------------------------------------------------------------------
-- Date:    9 September 1991
-- Version: 1.4
--
-- Filename: chip_pkg_body.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the package body that holds the
--              subroutines that can be accessed by any file. It currently
--              contains the Wired_And function to resolve the match bits
--              from each cell of the CAM array and the Wired_Or function
--              to resolve the output data and the validation data from
--              each cell.
-- Associated files: cam_cell_entity.vhd     : This file contains the entity
--                                             description of the CAM cell.
--                   cam_cell_structure.vhd  : This file contains the gate level
--                                             design of the CAM cell.
--                   chip_pkg.vhd            : This file is where the size of
--                                             the CAM array is defined. Other
--                                             declarations are also contained
--                                             in this file.
--                   cam_chip_entity.vhd     : This file contains the entity
--                                             description of the CAM chip.
--                   cam_chip_structure.vhd  : This file contains the structure
--                                             of the CAM chip. It is formed by
--                                             generating copies of the CAM
--                                             cell.
--                   chip_stimulus.vhd       : This file exercises the chip and
--                                             provides inputs to test the chip.
--                   chip_test_bench.vhd     : This file contains the test bench
--                                             for the CAM chip.
--                   chip_config.vhd         : This file contains the
--                                             configuration of the system.
--                   clock.vhd               : You guess!
--
-- History:  Version 1.2 (27 June 1991) - This is the first version.
--           Version 1.3 (3 July 1991) - added the function
--               Wired_Or_To_MVL7_Vector.
--           Version 1.4 (9 September 1991) - added the function
--               Wired_And_To_MVL7_Vector.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------

package body Chip_pkg is
```

```
-------------------------------------------------------------------------
--
-- The following functions are resolution functions. The signals coming into
-- these functions are resolved as follows: for the Wired_OR function, the
-- output is a '1' if one or more signals being resolved is a '1'. For the
-- Wired_And function, the output is a '0' if one or more signals being
-- resolved is a '0'. The Wired_Or code comes almost directly from the book
-- "VHDL: Hardware Description and Design" by Lipsett et al. and can be
-- found on pp 104-105.
--
-------------------------------------------------------------------------

        function Wired_Or (Input: Unconstrained_Vector) return MVL7 is
                variable Result: MVL7 := '0';
        begin
           for I in Input'Range loop
                if Input(I) = '1' then
                   Result := '1';
                   exit;
                elsif Input(I) = 'X' then
                   Result := 'X';
                else    --Input(I) = '0' or any other MVL7 value
                   null;
                end if;
           end loop;
           return Result;
        end Wired_Or;


        function Wired_And (Input: Unconstrained_Vector) return MVL7 is
                variable Result: MVL7 := '1';
        begin
           for I in Input'Range loop
                if Input(I) = '0' then
                   Result := '0';
                   exit;
                elsif Input(I) = 'X' then
                   Result := 'X';
                else    --Input(I) = '1' or any other MVL7 value
                   null;
                end if;
           end loop;
           return Result;
        end Wired_And;
```

```
-------------------------------------------------------------------------
-- The following function converts Wired_Or_Vector to MVL7_Vector type.
-------------------------------------------------------------------------

        function Wired_Or_To_MVL7_Vector (Input:  Wired_Or_Vector)
                                            return Vector_Word_length is
                variable Temp:  Vector_Word_length;
        begin
                for I in Word_length-1 downto 0 loop
                        Temp(I) := Input(I);
                end loop;
                return Temp;
        end Wired_Or_To_MVL7_Vector;

-------------------------------------------------------------------------
-- The following function converts Wired_And_Vector to MVL7_Vector type.
-------------------------------------------------------------------------

        function Wired_And_To_MVL7_Vector (Input:  Wired_And_Vector)
                                            return Vector_Depth is
                variable Temp:  Vector_Depth;
        begin
                for I in Depth-1 downto 0 loop
                        Temp(I) := Input(I);
                end loop;
                return Temp;
        end Wired_And_To_MVL7_Vector;


-------------------------------------------------------------------------
-- The following function converts Wired_And to MVL7 type.
-------------------------------------------------------------------------

        function Wired_And_To_MVL7 (Input:  Wired_And_Type) return MVL7 is
                variable Temp:  MVL7;
        begin
                Temp := Input;
                return Temp;
        end Wired_And_To_MVL7;


end Chip_pkg;
```

dual_phase_clock

```
------------------------------------------------------------
-- FILENAME: dual_phase_clock.vhd
--
-- DESCRIPTION: Entity description and architecture body of a
--              dual-phase clock generator (modified for ZYCAD).
--
-- APPLICABLE FILES: None
--
-- DESIGNER:  Gordon M. Kranz
--            Mark Mehalic
--
-- DEVELOPER: USAF
--
-- VERSION: 1.1
--
-- DATE: 14 May 91
--
------------------------------------------------------------
library ZYCAD;
use    ZYCAD.TYPES.all;
use    ZYCAD.COMPONENTS.all;
--
entity dual_phase_clock is
  generic (period : TIME);
  port   (PQ1    : inout MVL7 := '1';
          PQ2    : inout MVL7 := '0';
          RUN : in BOOLEAN := FALSE);
end dual_phase_clock;
--


architecture behavioral of dual_phase_clock is
--
 begin
  process(RUN, PQ1)
  begin
    if RUN and PQ1 = '0' then
      PQ1 <= transport '1' after period/2;
      PQ2 <= transport '0' after period/2;
    else if RUN and PQ1 = '1' then
         PQ1 <= transport '0' after period/2;
         PQ2 <= transport '1' after period/2;
       end if;
    end if;
  end process;
end behavioral;
```

Appendix F: Test Code Used to Test the MCC

This appendix contains the VHDL code used to test the MCC. The files contained in this appendix are the stimulus file used to input data into the MCC, the test bench, composed of the MCC and the stimulus, the configuration file for the test bench, and a test run of the MCC using these files. The times in the test run correspond directly to the stimulus file.

The Chip Stimulus

```
--------------------------------------------------------------------------
-- Date:    9 September 1991
-- Version: 2.0
--
-- Filename: chip_stimulus.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the stimulus to test the CAM chip.
-- Associated files: chip_pkg.vhd            : This file is where the size of
--                                             the CAM array is defined. Other
--                                             declarations are also contained
--                                             in this file.
--                   chip_pkg_body.vhd       : This file contains the sub-
--                                             routines Wired_And and Wired_Or
--                                             used by the chip.
--                   cam_chip_entity.vhd     : This file contains the entity
--                                             description of the CAM chip.
--                   cam_chip_structure.vhd  : This file contains the structure
--                                             of the CAM chip. It is formed by
--                                             generating copies of the CAM
--                                             cell.
--                   chip_test_bench.vhd     : This file contains the test bench
--                                              for the CAM chip.
--                   chip_config.vhd         : This file contains the
--                                              configuration of the system.
--                   clock.vhd               : You guess!
-- History:  Version 1.0 (10 June 1991)
--           Version 1.1 (12 June 1991) - changed M, RX, and RY to the actual
--               chip output ports T (match tag), R (data output), and
--               P (validation bit of data output).
--           Version 1.2 (25 June 1991) - changed inputs and outputs to vectors
--               (as opposed to matrices) to correspond to the actual inputs
```

```
--              and outputs of the chip. This change corresponds to Version
--              1.2 of cam_chip_entity and _structure.
--          Version 1.3 (3 July 1991) - modified times to correspond to the
--              change that the clock made.
--          Version 1.4 (16 July 1991) - changed data register to two separate
--              registers, one for the address and the other for data.
--          Version 1.5 (8 August 1991) - added ports for valid_out bit and
--              Resolved_Tags. These are preliminary changes to get ready to
--              go in and change what I have to what I really need.
--          Version 1.6 (28 August 1991) - deleted the bit select registers
--              and I will use only the MUXs. The MUXs will feed into the
--              Bit_Select_Bus.
--          Version 1.7 (30 August 1991) - In cam_chip_structure, deleted the
--              MUXs and replaced them with inverters since that is all they
--              were acting as. Also deleted registers to hold the Address_In,
--              Data_In, and Data_Out data and replaced them with buffers.
--              Thus, some of the timing of the stimulus had to be adjusted.
--          Version 2.0 (9 September 1991) - This version contains the
--              controller. As a consequence of adding the controller, a few
--              ports were added and some deleted.
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.Chip_pkg.all;

entity chip_stimulus is
        port(
                Data_In             : out Vector_Data_length;
                Address_In          : out Vector_Address_length;
                Read                : out MVL7;
                Write               : out MVL7;
                Data_Avail_MEM      : out MVL7;
                Data_Out            : out Vector_Word_length;
                Valid_Out           : out MVL7;
                Data_Out_Available  : out MVL7;
                Read_Miss           : out MVL7;
                Write_Miss          : out MVL7;
                Write_Hit           : out MVL7);
end chip_stimulus;


architecture behavior of chip_stimulus is

--------------------------------------------------------------------------------
-- This signal (Dummy) is a dummy signal to be used in
--  the sensitivity list for the process P1 below.
--------------------------------------------------------------------------------
        signal Dummy: MVL7;
```

```
-------------------------------------------------------------------------
--
-- The signal assignments below send data to the data buses in the chip which
--  send the inputs to each cell of the CAM array
-- The loops are used to send the data to the input ports more efficiently
--  than assigning the values one at a time.
--
--        1000 ns:  write '0' to each cell
--        2000 ns:  read contents of each cell
--        3000 ns:  search each cell for a '0'
--        4000 ns:  search each cell for a '1'
--        5000 ns:  write a '1' to each cell
--        6000 ns:  read contents of each cell
--        7000 ns:  search each cell for a '0'
--        8000 ns:  search each cell for a '1'
--
-------------------------------------------------------------------------

begin

   P1:
   process(Dummy)
        begin
-------------------------------------------------------------------------
-- The following data was used to test the chip using an address length of 3,
--  a data length of 3, and a depth of 3.
--  In order to test a chip of a different size, the following data must be
--  modified.
-------------------------------------------------------------------------

        -- Initialize the chip
        Data_In              <= transport "LLL";
        Address_In           <= transport "LLL";
        Read                 <= transport '0';
        Write                <= transport '0';
        Data_Avail_MEM       <= transport '0';
        Data_Out             <= transport "XXXXXX";
        Valid_Out            <= transport 'X';
        Data_Out_Available   <= transport '0' after 12 ns;
        Read_Miss            <= transport '0' after 3 ns;
        Write_Miss           <= transport '0' after 3 ns;
        Write_Hit            <= transport '0' after 3 ns;

        Data_Out             <= transport "000000" after 16 ns;
        Valid_Out            <= transport '0' after 19 ns;


        -- Read Miss
        Address_In           <= transport "001" after 39 ns;
```

```
Read            <= transport '1' after 39 ns;
Read_Miss       <= transport '1' after 58 ns;

Data_Avail_MEM  <= transport '1' after 57 ns;
Data_In         <= transport "001" after 57 ns;

Read            <= transport '0' after 65 ns;
Read_Miss       <= transport '0' after 68 ns;
Data_Avail_MEM   <= transport '0' after 65 ns;
Data_In         <= transport "LLL" after 88 ns;
Address_In      <= transport "LLL" after 88 ns;
                       -- address can't change until Sel_
                       -- Word_Sel is reset to '0'


-- Another Read Miss
Address_In      <= transport "010" after 1052 ns;
Read            <= transport '1' after 1052 ns;
Read_Miss       <= transport '1' after 1070 ns;

Data_Avail_MEM  <= transport '1' after 1070 ns;
Data_In         <= transport "010" after 1070 ns;

Read            <= transport '0' after 1078 ns;
Read_Miss       <= transport '0' after 1081 ns;
Data_Avail_MEM  <= transport '0' after 1078 ns;
Data_In         <= transport "LLL" after 1101 ns;
Address_In      <= transport "LLL" after 1101 ns;
                       -- address can't change until Sel_
                       -- Word_Sel is reset to '0'

-- Another Read Miss
Address_In      <= transport "011" after 1104 ns;
Read            <= transport '1' after 1104 ns;
Read_Miss       <= transport '1' after 1123 ns;

Data_Avail_MEM  <= transport '1' after 1122 ns;
Data_In         <= transport "011" after 1122 ns;

Read            <= transport '0' after 1130 ns;
Read_Miss       <= transport '0' after 1133 ns;
Data_Avail_MEM  <= transport '0' after 1130 ns;
Data_In         <= transport "LLL" after 1153 ns;
Address_In      <= transport "LLL" after 1153 ns;
                       -- address can't change until Sel_
                       -- Word_Sel is reset to '0'


-- Read Hit
Address_In      <= transport "001" after 1158 ns;
Read            <= transport '1' after 1160 ns;
```

```
Valid_out           <= transport '1' after 1219 ns;

Read                <= transport '0' after 1186 ns;
-- just to test to make sure this won't mess anything up
Read                <= transport '1' after 1300 ns;
Read                <= transport '0' after 1350 ns;

Data_Out            <= transport "000000" after 2081 ns;
Valid_out           <= transport '0' after 2084 ns;


-- Another Read Hit
Address_In          <= transport "010" after 2067 ns;
Read                <= transport '1' after 2067 ns;
Valid_out           <= transport '1' after 2127 ns;

Read                <= transport '0' after 2094 ns;

Data_Out            <= transport "000000" after 2999 ns;
Valid_out           <= transport '0' after 3002 ns;


-- Write Miss
Data_In             <= transport "000" after 3007 ns;
Address_In          <= transport "111" after 3007 ns;
Write               <= transport '1' after 3007 ns;

Write_Miss          <= transport '1' after 3027 ns;

Data_In             <= transport "LLL" after 3027 ns;
Address_In          <= transport "LLL" after 3027 ns;
Write               <= transport '0' after 3027 ns;

Write_Miss          <= transport '0' after 3030 ns;


-- Write Hit
Data_In             <= transport "111" after 3051 ns;
Address_In          <= transport "010" after 3051 ns;
Write               <= transport '1' after 3051 ns;

Write_Hit           <= transport '1' after 3070 ns;
-- Word_Select_Bus changes to select the word after 39 ns.
-- Write must go to '0' to clear the Word_Selector before the
-- *_In data changes. Otherwise an unwanted write will occur.
-- 31 ns after Write goes to '0', Word_Select_Bus changes to zeros.
-- Write can go low after being high for 20 ns.
Write               <= transport '0' after 3071 ns;
Data_In             <= transport "LLL" after 3098 ns;
--  Address_In      <= transport "LLL" after 3098 ns;
```

```
        Write_Hit           <= transport '0' after 3074 ns;


        -- Read Miss
        Address_In          <= transport "100" after 3099 ns;
        Read                <= transport '1' after 3099 ns;
        Read_Miss           <= transport '1' after 3118 ns;

        Data_Avail_MEM      <= transport '1' after 3118 ns;
        Data_In             <= transport "100" after 3118 ns;

        Read                <= transport '0' after 3126 ns;
        Read_Miss           <= transport '0' after 3129 ns;
        Data_Avail_MEM      <= transport '0' after 3126 ns;
        Data_In             <= transport "LLL" after 3149 ns;
        Address_In          <= transport "LLL" after 3149 ns;
                                -- address can't change until Sel_
                                -- Word_Sel output (WSR_Select)is
                                -- reset to '0'.


        -- Another Read Hit
        Address_In          <= transport "100" after 3154 ns;
        Read                <= transport '1' after 3154 ns;
        Valid_out           <= transport '1' after 3213 ns;

        Read                <= transport '0' after 3180 ns;

        Data_Out            <= transport "000000" after 4087 ns;
        Valid_out           <= transport '0' after 4090 ns;


    end process;

end behavior;
```

The Test Bench

```
---------------------------------- ----------------------------------------
-- Date:    9 September 1991
-- Version: 2.0
--
-- Filename: chip_test_bench.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file is the test bench to test the CAM chip.
-- Associated files: chip_pkg.vhd          : This file is where the size of
--                                           the CAM array is defined. Other
--                                           declarations are also contained
--                                           in this file.
--                   chip_pkg_body.vhd     : This file contains the sub-
--                                           routines Wired_And and Wired_Or
--                                           used by the chip.
--                   cam_chip_entity.vhd   : This file contains the entity
--                                           description of the CAM chip.
--                   cam_chip_structure.vhd : This file contains the structure
--                                           of the CAM chip. It is formed by
--                                           generating copies of the CAM
--                                           cell.
--                   chip_stimulus.vhd     : This file exercises the chip and
--                                           provides inputs to test the chip.
--                   chip_config.vhd       : This file contains the
--                                           configuration of the system.
--                   clock.vhd             : You guess!
--
-- History: Version 1.0 (10 June 1991)
--          Version 1.1 (12 June 1991) - changed M, RX, and RY to the actual
--              chip output ports T (match tag), R (data output), and
--              P (validation bit of data output).
--          Version 1.2 (25 June 1991) - changed input and output types to
--              correspond to Version 1.2 of cam_chip_entity and _structure.
--              The input and output registers are now those ports described
--              by test_chip below. Changed T to Tag, R to Data_Out, and
--              P to Valid_Out.
--          Version 1.3 (3 July 1991) - added Load port on test chip to allow
--              for a clock input.
--          Version 1.4 (16 July 1991) - changed ports to correspond to
--              Version 1.4 of cam_chip_entity and _structure.
--          Version 1.5 (8 August 1991) - added ports for valid_out bit and
--              Resolved_Tags. These are preliminary changes to get ready to
--              go in and change what I have to what I really need.
--          Version 1.6 (28 August 1991) - deleted the bit select registers
--              and I will use only the MUXs. The MUXs will feed into the
--              Bit_Select_Bus.
```

```
--          Version 1.7 (30 August 1991) - deleted the MUXs and replaced them
--              with inverters since that is all they were acting as.  Also
--              deleted registers to hold the Address_In, Data_In, and
--              Data_Out data and replaced them with buffers.
--          Version 2.0 (9 September 1991) - This version contains the
--              controller.  As a consequence of adding the controller, a few
--              ports were added and some deleted.
--
-- Author:  Curtis M. Winstead
--------------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.Chip_pkg.all;

entity chip_test_bench is
end chip_test_bench;

architecture structure of chip_test_bench is

-- These are the internal signals of the test bench
        signal Data_in                  : Vector_Data_length;
        signal Address_in               : Vector_Address_length;
        signal Read_in                  : MVL7;
        signal Write_in                 : MVL7;
        signal Data_Avail_MEM_in        : MVL7;
        signal Master_Reset             : MVL7;
        signal Data_out                 : Vector_Word_length;
        signal Data_stimulus_out        : Vector_Word_length;
        signal Valid_out                : MVL7;
        signal Valid_stimulus_out       : MVL7;
        signal Data_Avail_out           : MVL7;
        signal Data_Avail_stimulus_out  : MVL7;
        signal Read_Miss_out            : MVL7;
        signal Read_Miss_stimulus_out   : MVL7;
        signal Write_Miss_out           : MVL7;
        signal Write_Miss_stimulus_out  : MVL7;
        signal Write_Hit_out            : MVL7;
        signal Write_Hit_stimulus_out   : MVL7;

        signal CLK                      : MVL7;
        signal CLKnot                   : MVL7;
        signal RUN                      : boolean := FALSE;
        signal stop_sim                 : boolean := FALSE;

-- This is the chip under the test
        component test_chip
                port(
                        Data_In         : in Vector_Data_length;
                        Address_In      : in Vector_Address_length;
                        Read            : in MVL7;
                        Write           : in MVL7;
```

```
                Data_Avail_MEM      : in MVL7;
                Master_Reset        : in MVL7;
                CP                  : in MVL7;
                CPnot               : in MVL7;
                Data_Out            : out Vector_Word_length;
                Valid_Out           : out MVL7;
                Data_Out_Available  : out MVL7;
                Read_Miss           : out MVL7;
                Write_Miss          : out MVL7;
                Write_Hit           : out MVL7);
        end component;

-- This is used to compare for correct output
        component stimulus
                port(
                Data_In             : out Vector_Data_length;
                Address_In          : out Vector_Address_length;
                Read                : out MVL7;
                Write               : out MVL7;
                Data_Avail_MEM      : out MVL7;
                Data_Out            : out Vector_Word_length;
                Valid_Out           : out MVL7;
                Data_Out_Available  : out MVL7;
                Read_Miss           : out MVL7;
                Write_Miss          : out MVL7;
                Write_Hit           : out MVL7);
        end component;


        component dual_phase_clock
          generic (period : TIME := 34 ns);
          port   (PQ1  : inout MVL7 := '1';
                  PQ2  : inout MVL7 := '0';
                  RUN : in BOOLEAN := FALSE);
        end component;


begin
        Chip: test_chip
                port map(
                Data_In             => Data_in,
                Address_In          => Address_in,
                Read                => Read_in,
                Write               => Write_in,
                Data_Avail_MEM      => Data_Avail_MEM_in,
                Master_Reset        => Master_Reset,
                CP                  => CLK,
                CPnot               => CLKnot,
                Data_Out            => Data_out,
                Valid_Out           => Valid_out,
                Data_Out_Available  => Data_Avail_out,
```

```
                Read_Miss           => Read_Miss_out,
                Write_Miss          => Write_Miss_out,
                Write_Hit           => Write_Hit_out);


        Generator: stimulus
                port map(
                        Data_In             => Data_in,
                        Address_In          => Address_in,
                        Read                => Read_in,
                        Write               => Write_in,
                        Data_Avail_MEM              => Data_Avail_MEM_in,
                        Data_Out            => Data_stimulus_out,
                        Valid_Out           => Valid_stimulus_out,
                        Data_Out_Available  => Data_Avail_stimulus_out,
                        Read_Miss           => Read_Miss_stimulus_out,
                        Write_Miss          => Write_Miss_stimulus_out,
                        Write_Hit           => Write_Hit_stimulus_out);


        CLOCK_IN: dual_phase_clock
                port map(CLK,
                        CLKnot,
                        RUN);


Master_Reset <= '1',            -- puts chip into initial state
                '0' after 38 ns;


RUN <= TRUE;


-------------------------------------------------------------------------
-- The following process checks for correct output.
-------------------------------------------------------------------------
        ERROR_TEST: process(Valid_out, Read_Miss_out,
                        Write_Miss_out, Write_Hit_out)
          begin
                assert (Valid_out     = Valid_stimulus_out and
                        Read_Miss_out  = Read_Miss_stimulus_out and
                        Write_Miss_out = Write_Miss_stimulus_out and
                        Write_Hit_out  = Write_Hit_stimulus_out)
                report "FAILED TEST"
                severity warning;
        end process ERROR_TEST;

-- Stops simulation after 30000 ns
   stop_sim <= TRUE after 30000 ns;
```

```
-------------------------------------------------------------------------
-- This process stops the simulation when stop_sim is true.
-------------------------------------------------------------------------
STOP_CONTROL: process
        begin
        wait until stop_sim = TRUE;
        assert false report "Simulation Done."
                severity failure;
end process STOP_CONTROL;


end structure;
```

The Configuration File

```
------------------------------------------------------------------------
-- Date:    28 March 1991
-- Version:  1.0
--
-- Filename:  chip_config.vhd
-- System:    ZYCAD, VLSI net
-- Language:  VHDL
--
-- Description:  This is the configuration specification file for the CAM
--           chip test bench.
-- Associated files:  chip_pkg.vhd          : This file is where the size of
--                                            the CAM array is defined.  Other
--                                            declarations are also contained
--                                            in this file.
--                    chip_pkg_body.vhd     : This file contains the sub-
--                                            routines Wired_And and Wired_Or
--                                            used by the chip.
--                    cam_chip_entity.vhd   : This file contains the entity
--                                            description of the CAM chip.
--                    cam_chip_structure.vhd: This file contains the structure
--                                            of the CAM chip.  It is formed by
--                                            generating copies of the CAM
--                                            cell.
--                    chip_stimulus.vhd     : This file exercises the chip and
--                                            provides inputs to test the chip.
--                    chip_test_bench.vhd   : This file contains the test bench
--                                            for the CAM chip.
--                    clock.vhd             : You guess!
-- History:
-- Author:  Curtis M. Winstead
------------------------------------------------------------------------

use work.all;

configuration chip_config of chip_test_bench is

for structure

        for Chip:  test_chip use entity work.CAM_chip(structure);
        end for;

        for Generator:  stimulus use entity work.chip_stimulus(behavior);
        end for;

end for;

end chip_config;
```

Test Run

```
# 0 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "XXXXXX")
        M1:      ACTIVE  /CHIP_TEST_BENCH/DATA_STIMULUS_OUT  (value  =
"XXXXXX")
   M16:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_MEM_IN (value = '0')
   M15:  ACTIVE /CHIP_TEST_BENCH/WRITE_IN (value = '0')
   M14:  ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '0')
   M13:  ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "LLL")
   M12:  ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "LLL")
    M2:  ACTIVE /CHIP_TEST_BENCH/RES_SIG_VALID_OUT (value = 'X')
    M3:  ACTIVE /CHIP_TEST_BENCH/RSV_STIMULUS_OUT (value = 'X')
3 NS
   M10:  ACTIVE /CHIP_TEST_BENCH/WRITE_HIT_OUT (value = '0')
    M8:  ACTIVE /CHIP_TEST_BENCH/WRITE_MISS_OUT (value = '0')
    M6:  ACTIVE /CHIP_TEST_BENCH/READ_MISS_OUT (value = '0')
    M11:    ACTIVE /CHIP_TEST_BENCH/WRITE_HIT_STIMULUS_OUT (value =
'0')
     M9:    ACTIVE /CHIP_TEST_BENCH/WRITE_MISS_STIMULUS_OUT (value =
'0')
     M7:    ACTIVE /CHIP_TEST_BENCH/READ_MISS_STIMULUS_OUT (value =
'0')
12 NS
    M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
    M5:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_STIMULUS_OUT (value =
'0')
16 NS
        M1:      ACTIVE  /CHIP_TEST_BENCH/DATA_STIMULUS_OUT  (value  =
"000000")
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "000000")
19 NS
    M3:  ACTIVE /CHIP_TEST_BENCH/RSV_STIMULUS_OUT (value = '0')
    M2:  ACTIVE /CHIP_TEST_BENCH/RES_SIG_VALID_OUT (value = '0')

1070 NS
    M6:  ACTIVE /CHIP_TEST_BENCH/READ_MISS_OUT (value = '1')
   M12:  ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "010")
   M16:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_MEM_IN (value = '1')
     M7:    ACTIVE /CHIP_TEST_BENCH/READ_MISS_STIMULUS_OUT (value =
'1')
1078 NS
   M16:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_MEM_IN (value = '0')
   M14:  ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '0')
1081 NS
    M6:  ACTIVE /CHIP_TEST_BENCH/READ_MISS_OUT (value = '0')
     M7:    ACTIVE /CHIP_TEST_BENCH/READ_MISS_STIMULUS_OUT (value =
'0')
1101 NS
   M13:  ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "LLL")
   M12:  ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "LLL")
```

```
1104 NS
   M14:   ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '1')
   M13:   ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "011")
1122 NS
   M12:   ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "011")
   M16:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_MEM_IN (value = '1')
1123 NS
    M6:   ACTIVE /CHIP_TEST_BENCH/READ_MISS_OUT (value = '1')
     M7:    ACTIVE /CHIP_TEST_BENCH/READ_MISS_STIMULUS_OUT (value =
'1')
1130 NS
   M16:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_MEM_IN (value = '0')
   M14:   ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '0')
1133 NS
    M6:   ACTIVE /CHIP_TEST_BENCH/READ_MISS_OUT (value = '0')
     M7:    ACTIVE /CHIP_TEST_BENCH/READ_MISS_STIMULUS_OUT (value =
'0')
1153 NS
   M13:   ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "LLL")
   M12.   ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "LLL")
1158 NS
   M13:   ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "001")
1160 NS
   M14:   ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '1')
1186 NS
   M14:   ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '0')
1216 NS
    M:   ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
1219 NS
    M3:   ACTIVE /CHIP_TEST_BENCH/RSV_STIMULUS_OUT (value = '1')
    M2:   ACTIVE /CHIP_TEST_BENCH/RES_SIG_VALID_OUT (value = '1')
1223 NS
    M4:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1228 NS
    M4:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1252 NS
    M:   ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1259 NS
    M4:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1264 NS
    M4:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1286 NS
    M:   ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
1293 NS
    M4:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1298 NS
    M4:   ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1300 NS
   M14:   ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '1')
1320 NS
    M:   ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
```

```
1327 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1332 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1350 NS
  M14:  ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '0')
1354 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1361 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1366 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1388 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
1395 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1400 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1422 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
1429 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1434 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1456 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1463 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1468 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1490 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
1497 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1502 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1524 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
1531 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1536 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1558 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1565 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1570 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1592 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
```

```
1599 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1604 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1626 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
1633 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1638 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1660 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1667 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1672 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1694 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
1701 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1706 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1728 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
1735 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1740 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1762 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1769 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1774 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1796 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
1803 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1808 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1830 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
1837 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1842 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1864 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1871 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
```

```
1876 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1898 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
1905 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1910 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1932 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
1939 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1944 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
1966 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
1973 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
1978 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
2000 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "011011")
2007 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
2012 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
2034 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "001001")
2041 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '1')
2046 NS
   M4:  ACTIVE /CHIP_TEST_BENCH/DATA_AVAIL_OUT (value = '0')
2065 NS
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "010010")
2067 NS
  M14:  ACTIVE /CHIP_TEST_BENCH/READ_IN (value = '1')
  M13:  ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "010")
2081 NS
       M1:      ACTIVE  /CHIP_TEST_BENCH/DATA_STIMULUS_OUT  (value =
"000000")
    M:  ACTIVE /CHIP_TEST_BENCH/DATA_OUT (value = "000000")
2084 NS
   M3:  ACTIVE /CHIP_TEST_BENCH/RSV_STIMULUS_OUT (value = '0')
   M2:  ACTIVE /CHIP_TEST_BENCH/RES_SIG_VALID_OUT (value = '0')

3007 NS
  M15:  ACTIVE /CHIP_TEST_BENCH/WRITE_IN (value = '1')
  M13:  ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "111")
  M12:  ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "000")
```

```
3027 NS
    M8:   ACTIVE /CHIP_TEST_BENCH/WRITE_MISS_OUT (value = '1')
   M15:   ACTIVE /CHIP_TEST_BENCH/WRITE_IN (value = '0')
   M13:   ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "LLL")
   M12:   ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "LLL")
     M9:    ACTIVE /CHIP_TEST_BENCH/WRITE_MISS_STIMULUS_OUT (value =
'1')
3030 NS
    M8:   ACTIVE /CHIP_TEST_BENCH/WRITE_MISS_OUT (value = '0')
     M9:    ACTIVE /CHIP_TEST_BENCH/WRITE_MISS_STIMULUS_OUT (value =
'0')
3051 NS
   M15:   ACTIVE /CHIP_TEST_BENCH/WRITE_IN (value = '1')
   M13:   ACTIVE /CHIP_TEST_BENCH/ADDRESS_IN (value = "010")
   M12:   ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "111")
3054 NS
    M8:   ACTIVE /CHIP_TEST_BENCH/WRITE_MISS_OUT (value = '0')
3070 NS
   M10:   ACTIVE /CHIP_TEST_BENCH/WRITE_HIT_OUT (value = '1')
    M11:     ACTIVE /CHIP_TEST_BENCH/WRITE_HIT_STIMULUS_OUT (value =
'1')
3071 NS
   M15:   ACTIVE /CHIP_TEST_BENCH/WRITE_IN (value = '0')
3074 NS
   M10:   ACTIVE /CHIP_TEST_BENCH/WRITE_HIT_OUT (value = '0')
    M11:     ACTIVE /CHIP_TEST_BENCH/WRITE_HIT_STIMULUS_OUT (value =
'0')
3098 NS
   M12:   ACTIVE /CHIP_TEST_BENCH/DATA_IN (value = "LLL")

30000 NS
Assertion FAILURE at 30000 NS in design unit STRUCTURE from process
/CHIP_TEST_BENCH/STOP_CONTROL:
  "Simulation Done."
```

Appendix G: The VHDL Code of Simple Memory System

This appendix contains the VHDL code of the simple memory system used to thoroughly test the MCC. It contains the code for a simple behavioral description of the CPU, the main memory, and the structural description of the memory system consisting of the CPU, main memory, and the MCC. The package containing the MEM_Delay constant and a function to convert hexadecimal values to MVL7 vectors is also included.

The CPU

```
--------------------------------------------------------------------------
-- Date: 28 October 1991
-- Version: 1.0
--
-- Filename: cpu.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains a simple behavioral model of a CPU. Its
--              main purpose is to generate addresses to the bus with a
--              read or write request.
-- Associated files: chip_pkg.vhd          : This file contains declarations
--                                           for constants, types, functions,
--                                           etc.
--                   mem_sys_pkg.vhd       : This file contains declarations
--                                           for constants, types, functions,
--                                           etc. specific to the memory
--                                           system.
--                   mem_sys_pkg_body.vhd  : This file contains the function
--                                           to convert hex numbers into
--                                           MVL7_Vector.
--
-- History:
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------

library ZYCAD;
use ZYCAD.types.all;
use WORK.Chip_pkg.all;
use WORK.mem_sys_pkg.all;
use STD.TEXTIO.all;
```

```
-------------------------------------------------------------------------
-- This is the entity declaration of the CPU.
-------------------------------------------------------------------------
entity CPU is
        port(   Read_port               : inout MVL7 := '0';
                Write_port              : inout MVL7 := '0';
                MCC_Prefetching         : in MVL7 := '0';
                Address                 : out Vector_Address_length :=
                                (Word_length-1 downto Data_length => 'L'));
end CPU;


-------------------------------------------------------------------------
-- This is the architecture description of the CPU.
-------------------------------------------------------------------------
architecture CPU of CPU is
begin


        ---------------------------------------------------------------
        -- This process generates a Read and Write request on every other
        --  100 ns time block.
        ---------------------------------------------------------------
        RW:
        process
        begin

                wait for 100 ns;
                RWLoop: loop
                        Read_port <= '1';
                        wait for 30 ns;
                        Read_port <= '0';
                        wait for 70 ns;
                        ------------------------------------------------
                        -- This line blocks the CPU during prefetching
                        --  cycle.
                        ------------------------------------------------
                        if MCC_Prefetching = '1' then
                                wait on MCC_Prefetching until
                                                MCC_Prefetching = '0';
                                wait for 10 ns;
                        end if;
                        Write_port <= '1';
                        wait for 20 ns;
                        Write_port <= '0';
                        wait for 80 ns;
                end loop RWLoop;

        end process RW;
```

```
        ------------------------------------------------------------
        -- This process reads the address from the file "cpu.dat" and
        -- converts the hex number into an MVL7_Vector to be supplied to
        -- the bus.
        ------------------------------------------------------------
        Read_Address:
        process(Read_port, Write_port)
                file Input_file: TEXT is in "cpu.dat";
                variable Addr: String(8 downto 1);
                variable L1: Line;
        begin
                if (Read_port = '1' and not Read_port'Stable) or
                        (Write_port = '1' and not Write_port'Stable) then
                        assert not Endfile(Input_file) report -- if end of
                                "No more addresses."            -- file then
                        severity failure;                       -- simulation
                                                                -- quits
                        Readline (Input_file, L1);
                        Read(L1, Addr);
                        Address <= transport HEX_To_MVL7V(Addr);
                end if;

                ------------------------------------------------------
                -- Puts 'L's on Address port after CPU stops its request.
                ------------------------------------------------------
                if (Read_port = '0' and not Read_port'Stable) or
                        (Write_port = '0' and not Write_port'Stable) then
                        for J in Word_length-1 downto Data_length loop
                                Address(J) <= 'L' after 30 ns;
                        end loop;
                end if;

        end process;

end CPU;


------------------------------------------------------------------------
-- The following is the configuration file for the CPU.  I put it here
-- so as not to clutter up the directory.  It was used to test the CPU
-- only, before being put into the memory system.
------------------------------------------------------------------------
use work.all;
configuration cpu_config of CPU is

for CPU
end for;

end cpu_config;
```

Main Memory

```
--------------------------------------------------------------------------
-- Date: 28 October 1991
-- Version: 1.0
--
-- Filename: mem.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains a simple behavioral model of a memory. Its
--              main purpose is to generate data to the bus when the CPU
--              requests a write or when a read miss occurs on the MCC. I
--              realize this is not the true function of memory, but in order
--              to simplify the testing of the CAM chip, it generates the
--              address on a write request. The MCC doesn't know any better
--              and that's what is being tested anyway.
-- Associated files: chip_pkg.vhd          : This file contains declarations
--                                           for constants, types, functions,
--                                           etc.
--                   mem_sys_pkg.vhd       : This file contains declarations
--                                           for constants, types, functions,
--                                           etc. specific to the memory
--                                           system.
--                   mem_sys_pkg_body.vhd  : This file contains the function
--                                           to convert hex numbers into
--                                           MVL7_Vector.
--
-- History:
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.mem_sys_pkg.all;
use WORK.Chip_pkg.all;
use STD.TEXTIO.all;


--------------------------------------------------------------------------
-- This is the entity declaration of the MEM(ory).
--------------------------------------------------------------------------
entity MEM is
        port(   CPU_Write    : in MVL7;
                MCC_Miss     : in MVL7;
                Data         : out Vector_Data_length :=
                               (Data_length-1 downto 0 => 'L');
                Data_Avail   : out MVL7 := '0');
end MEM;
```

```
-------------------------------------------------------------------------
-- This is the architecture description of the MEM(ory).
-------------------------------------------------------------------------
architecture MEM of MEM is
begin


        -------------------------------------------------------------
        -- This process reads data from the "mem.dat" file, converts it from
        -- hex to MVL7_Vector, and supplies it to the data port.
        -- It reads data when the CPU requests a write and when the MCC
        -- has a read miss.
        -------------------------------------------------------------
        Read_Data:
        process
                file Input_file:  TEXT is in "mem.dat";
                variable Dat:  String(8 downto 1);
                variable L1: Line;

        begin

                wait on CPU_Write, MCC_Miss;

                if CPU_Write = '1' or MCC_Miss = '1' then
                        if (CPU_Write = '1' and not CPU_Write'Stable) or
                                (MCC_Miss = '1' and not MCC_Miss'Stable) then
                                assert not Endfile(Input_file) report
                                        "No more data." -- if end of file then
                                severity failure;       -- simuluation quits

                                if MCC_Miss = '1' then
                                        wait for MEM_Delay;
                                end if;
                                Readline (Input_file, L1);
                                Read(L1, Dat);
                                Data <= transport HEX_To_MVL7V(Dat);
                        end if;


                        ---------------------------------------------------
                        -- This if statement synchronizes the address and
                        -- data going to 'L's after a write request.
                        ---------------------------------------------------
                        if CPU_Write = '1' then
                                wait for 50 ns;
                        else
                                wait for 41 ns;
                        end if;
```

```
                    ------------------------------------------------
                    -- Puts 'L's on data port at the same time the
                    --  the address gets all 'L's.
                    ------------------------------------------------
                    for J in Data_length-1 downto 0 loop
                            Data(J) <= 'L';
                    end loop;

            end if;

    end process;


    ------------------------------------------------------------------
    -- This block is used to assert the Data_Avail_MEM port of the MCC
    --  when a read miss occurs.
    ------------------------------------------------------------------
    Data_Avail_MEM:
    block(MCC_Miss = '1' and not MCC_Miss'Stable)
    begin
      process
      begin
            wait on guard;
            Data_Avail <= '1';
            wait for 8 ns;
            Data_Avail <= '0';
      end process;
    end block Data_Avail_MEM;

end MEM;
```

The Memory System

```
--------------------------------------------------------------------------
-- Date: 28 October 1991
-- Version: 1.0
--
-- Filename: mem_system.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the structure of the memory system. It
--              contains the CPU, MEM, and the MCC. The CPU supplies the
--              function requests, MEM supplies the data, and the MCC is
--              therefore exercised.
-- Associated files: chip_pkg.vhd    : This file contains declarations for
--                                     constants, types, functions, etc.
--                   cpu.vhd         : This file contains a simple behavioral
--                                     description of a CPU. Its main function
--                                     is to supply addresses and functions
--                                     requests.
--                   mem.vhd         : This file contains a simple behavioral
--                                     description of main memory. Its main
--                                     funtions is to supply data to the system.
-- History:
--
-- Author: Curtis M. Winstead
--------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;
use WORK.Chip_pkg.all;

--------------------------------------------------------------------------
-- This is the entity description of the memory system.
--------------------------------------------------------------------------
entity mem_system is
        port(   Master_Reset        : in MVL7 := '0';
                CP                  : in MVL7 := '1';
                CPnot               : in MVL7 := '0';
                Data_Out            : out Vector_Word_length;
                Valid_Out           : out MVL7;
                Data_Out_Available  : out MVL7;
                Write_Miss          : out MVL7;
                Write_Hit           : out MVL7;
                Read_Miss           : inout MVL7);
end mem_system;


--------------------------------------------------------------------------
-- This is the structural description of the memory system.
--------------------------------------------------------------------------
architecture structure of mem_system is
```

```
        -- Internal signals used
        signal Write_Sig       : MVL7;
        signal Read_Sig        : MVL7;
        signal Address_Sig     : Vector_Address_length;
        signal Data_Sig        : Vector_Data_length;
        signal Data_Avail      : MVL7;
        signal Miss            : MVL7;
        signal Valid_Out_Sig   : MVL7;


        -- Components used

        component CPU
                port(   Read_port           : inout MVL7;
                        Write_port          : inout MVL7;
                        MCC_Prefetching     : in MVL7;
                        Address             : out Vector_Address_length);
        end component;

        component MEM
                port(   CPU_Write   : in MVL7;
                        MCC_Miss    : in MVL7;
                        Data        : out Vector_Data_length;
                        Data_Avail  : out MVL7);
        end component;

        component cam_chip
                port(   Data_In             : in Vector_Data_length;
                        Address_In          : in Vector_Address_length;
                        Read                : in MVL7;
                        Write               : in MVL7;
                        Data_Avail_MEM      : in MVL7;
                        Master_Reset        : in MVL7;
                        CP                  : in MVL7;
                        CPnot               : in MVL7;
                        Data_Out            : out Vector_Word_length;
                        Valid_Out           : out MVL7;
                        Data_Out_Available  : out MVL7;
                        Read_Miss           : out MVL7;
                        Write_Miss          : out MVL7;
                        Write_Hit           : out MVL7);
        end component;

begin

        Address_Generator: CPU
                port map(Read_port          => Read_Sig,
                        Write_port          => Write_Sig,
                        MCC_Prefetching     => Valid_Out_Sig,
                        Address             => Address_Sig);
```

```
        Data_Generator: MEM
                port map(CPU_Write => Write_Sig,
                        MCC_Miss   => Miss,
                        Data       => Data_Sig,
                        Data_Avail => Data_Avail);

        MCC: cam_chip
                port map(Data_In           => Data_Sig,
                        Address_In         => Address_Sig,
                        Read               => Read_Sig,
                        Write              => Write_Sig,
                        Data_Avail_MEM     => Data_Avail,
                        Master_Reset       => Master_Reset,
                        CP                 => CP,
                        CPnot              => CPnot,
                        Data_Out           => Data_Out,
                        Valid_Out          => Valid_Out_Sig,
                        Data_Out_Available => Data_Out_Available,
                        Read_Miss          => Read_Miss,
                        Write_Miss         => Write_Miss,
                        Write_Hit          => Write_Hit);

        -- Signal assignments for output ports of the system
        Miss     <= Read_Miss;
        Valid_Out <= Valid_Out_Sig;

end structure;
```

The Memory System Package Declaration

```
-------------------------------------------------------------------------
-- Date: 24 October 1991
-- Version: 1.0
--
-- Filename: mem_sys_pkg.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the function declaration of the function
--              that converts hex values to MVL7_Vector and constants.
-- Associated files:   cpu.vhd    : This file contains a simple behavioral
--                                  description of a CPU. Its main function
--                                  is to supply addresses and functions
--                                  requests.
--                     mem.vhd    : This file contains a simple behavioral
--                                  description of main memory. Its main
--                                  function is to supply data to the system.
--
-- History:
--
-- Author: Curtis M. Winstead
-------------------------------------------------------------------------
library ZYCAD;
use ZYCAD.types.all;

package mem_sys_pkg is

        ---------------------------------------------------------------
        -- This is the funtion declaration.
        ---------------------------------------------------------------
        function HEX_To_MVL7V (Input: String) return MVL7_Vector;


        ---------------------------------------------------------------
        -- This constant is the time it takes main memory to respond to
        --  a read.
        ---------------------------------------------------------------
        constant MEM_Delay: Time := 20 ns;

end mem_sys_pkg;
```

The Memory System Package Body

```
-------------------------------------------------------------------------
-- Date: 24 October 1991
-- Version: 1.0
--
-- Filename: mem_sys_pkg_body.vhd
-- System:   ZYCAD, VLSI net
-- Language: VHDL
--
-- Description: This file contains the function that converts hex values
--              to MVL7_Vector.
-- Associated files:   mem_sys_pkg.vhd : This file contains the funtion
--                                       declaration for this file.
--                     cpu.vhd         : This file contains a simple
--                                       behavioral description of a CPU.
--                                       Its main function is to supply
--                                       addresses and functions requests.
--                     mem.vhd         : This file contains a simple
--                                       behavioral description of main
--                                       memory.  Its main function is to
--                                       supply data to the system.
--
-- History:
--
-- Author: Curtis M. Winstead
-------------------------------------------------------------------------
package body mem_sys_pkg is


        ---------------------------------------------------------------
        -- This function converts hex values to MVL7_Vector.
        ---------------------------------------------------------------
        function HEX_To_MVL7V (Input: String) return MVL7_Vector is
                variable Address_Vector: MVL7_Vector(31 downto 0)
                                        := (31 downto 0 => '0');
                variable char: Character;
        begin
                for I in 8 downto 1 loop
                        char := Input(I);
                        case char is
                                when '0' => Address_Vector(4*I-1 downto 4*I-4)
                                                        := "0000";
                                when '1' => Address_Vector(4*I-1 downto 4*I-4)
                                                        := "0001";
                                when '2' => Address_Vector(4*I-1 downto 4*I-4)
                                                        := "0010";
                                when '3' => Address_Vector(4*I-1 downto 4*I-4)
                                                        := "0011";
                                when '4' => Address_Vector(4*I-1 downto 4*I-4)
                                                        := "0100";
```

```
                    when '5' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "0101";
                    when '6' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "0110";
                    when '7' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "0111";
                    when '8' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1000";
                    when '9' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1001";
                    when 'A' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1010";
                    when 'B' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1011";
                    when 'C' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1100";
                    when 'D' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1101";
                    when 'E' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1110";
                    when 'F' => Address_Vector(4*I-1 downto 4*I-4)
                                                    := "1111";
                    when others => null;
                end case;
            end loop;

            return Address_Vector;

        end HEX_To_MVL7V;

end mem_sys_pkg;
```

Bibliography

1. Ackland, B. D. "A Bit-Slice Controller," *The 6th Annual Symposium on Computer Architecture, 6*: 75-82 (April 1979).

2. Chisvin, Lawrence and R. James Duckworth. "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *Computer,* 51-64 (July 1989).

3. DeCegama, Angel L. *The Technology of Parallel Processing; Parallel Processing Architectures and VLSI Hardware* , Volume 1. Englewood Cliffs NJ: Prentice Hall, 1989.

4. Deitel, Harvey M. *Operating Systems* (Second Edition). Reading MA: Addison-Wesley Publishing Company, 1990.

5. Goksel, A. Kemal *et al.* "A Content Addressable Memory Management Unit with On-Chip Data Cache," *IEEE Journal of Solid-State Circuits, 24*: 592-595 (June 1989).

6. Hanlon, A. G. "Content-Addressable and Associative Memory Systems: A Survey," *IEEE Transactions on Electronic Computers, EC-15*: 509-521 (August 1966).

7. Hayes, John P. *Computer Architecture and Organization* (Second Edition). New York: McGraw-Hill Book Company, 1988.

8. Herrmann, Frederick P. *et al.* "A Dynamic Three-State Memory Cell for High-Density Associative Processors," *IEEE Journal of Solid-State Circuits, 26*: 537-541 (April 1991).

9. Hobart, William C., Jr. *An Investigation of the Locality of Memory Accesses During Symbolic Program Execution.* PhD dissertation. The University of Texas at Austin, 1989.

10. Jones, S. "Design, Selection and Implementation of a Content-Addressable Memory for a VLSI CMOS Chip Architecture," *IEEE Proceedings, 135*: 165-172 (May 1988).

11. Lipsett, Roger *et al. VHDL: Hardware Description and Design* . Boston: Kluwer Academic Publishers, 1990.

12. Mano, M. Morris. *Digital Logic and Computer Design* . Englewood Cliffs NJ: Prentice-Hall, Inc., 1979.

13. McAuley, Anthony J. *et al.* "A Self-Testing Reconfigurable CAM," *IEEE Journal of Solid-State Circuits, 26*: 257-261 (March 1991).

14. Mehalic, Capt Mark A., Falculty, Air Force Institute of Technology. Personal interview. Wright-Patterson AFB OH, 6 June through 30 September 1991.

15. Minker, Jack. "An Overview of Associative or Content-Addressable Memory Systems and a KWIC Index to the Literature," *Computing Reviews, 12*: 453-504 (October 1971).

16. Ogura, Takeshi. "A 20-kbit Associative Memory LSI for Artificial Intelligence Machines," *IEEE Journal of Solid-State Circuits, 24*: 1014-1020 (August 1989).

17. Quinones, Lisa K. "The NS32605 Cache Controller," *IEEE Spring COMPCON*, 218-222 (1988).

18. Rowe, Captain Mark C. *Content-Addressable Memory Manager; Design and Evaluation*. MS thesis, AFIT-GE/ENG/85D-36. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985 (AD-A164037).

19. Shinn, Captain Wook H. *Design and Implementation of High Performance Content-Addressable Memories*. MS thesis, AFIT-GE/ENG/85D-39. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985 (AD-A163995).

20. Wade, Jon P. "A Ternary Content Addressable Search Engine," *IEEE Journal of Solid-State Circuits, 24*: 1003-1013 (August 1989).

21. Weste, Neil H. E. and Kamran Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective* . Reading, MA: Addison-Wesley Publishing Company, 1985.

22. *ZYCAD System VHDL User's Manual*, ZYCAD Corp., 1990.